# Agenttipohjaiset menetelmät MES-järjestelmän toteutuksessa

Mikko Rahikainen

**Perustieteiden korkeakoulu**

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 27.11.2017.

**Työn valvoja:**

Prof. Kary Främling

Aalto-yliopisto
Perustieteiden korkeakoulu

AALTO-YLIOPISTO
PERUSTIETEIDEN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Mikko Rahikainen

Työn nimi: Agenttipohjaiset menetelmät MES-järjestelmän toteutuksessa

Päivämäärä: 27.11.2017 Kieli: Suomi Sivumäärä: 7+55

Tietotekniikan laitos

Professuuri: Ohjelmistotekniikka

Työn valvoja ja ohjaaja: Prof. Kary Främling

Visio tulevaisuuden tuotannosta on modulaariset ja tehokkaat tuotantojärjestelmät, joissa tuotteet ohjaavat omaa tuotantoprosessiaan. Näin on tarkoitus toteuttaa yksilöllisten tuotteiden tuotanto yhden tuotteen eräkoolla ilman että tarvitsee luopua massatuotannon taloudellisista eduista. Termi "Industry 4.0"kuvaa tätä suunniteltua neljättä teollista vallankumousta.
Agenttipohjaisten järjestelmien toimivuus ympäristöissä joissa ei voida käyttää keskitettyä arkkitehtuuria päätöksen tekoon, tai toimivuus ympäristöissä joissa vaaditaan nopeaa vasteaikaa ja luotettavuutta, vastaavat osin näihin haasteisiin.
Työssä tutkittiin MES-näkökulmasta hienokuormituksen toteutusta agenttipohjaisilla menetelmillä ja rakennettiin JADE-ohjelmistokehystä käyttäen tuotantosimulaatio kokeellista tutkimusta varten.
Työssä toteutettiin kaksi erilaista tapaa tuotannon hienokuormitukseen agenttipohjaisesti. Ensimmäiseksi toteutettiin markkinamekanismiin perustuva hienokuormitustapa, jossa tilausagentit päättävät resurssiagenttien käytöstä saamiensa tarjouksien perusteella. Toiseksi toteutettiin geneettiseen algoritmiin pohjautuva tapa, jossa optimointiagentti aikatauluttaa ajettavat työt resurssiagenteille.

Avainsanat: Industry 4.0, valmistuksenohjaus, hienokuormitus, MES, agentti, DAI, MAS

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Mikko Rahikainen

Title: Agent Technologies in MES Implementation

Date: 27.11.2017 Language: Finnish Number of pages: 7+55

Department of Computer Science

Professorship: Software Systems and Technologies

Supervisor: Prof. Kary Främling

Advisor: Prof. Kary Främling

Modular and efficient production systems where the products control their own production process is the vision for future. These systems are supposed to carry out the production of individual products with a batch size of one without having to abandon the benefits of mass production. The term "Industry 4.0" is for this planned fourth industrial revolution.
Multi agent systems work in environments where a centralized architecture can not be used or make decisions in environments requiring fast responsiveness and reliability. These attributes align with Industry 4.0 requirements.
In this thesis the implementation of agent-based scheduling is studied and an implementation is built using JADE-framework. The implementation is used to simulate production for research purposes.
Two different algorithms for agent based detailed scheduling are implemented. A contract net based algorithm is crafted so that order agents and resource agents schedule production based on bidding and pricing mechanism. Secondly a genetic algorithm with an optimization agent is implemented to schedule tasks for the resource agents.

Keywords: Industry 4.0, manufacturing operations management, detailed scheduling, MES, agent, DAI, MAS

# Esipuhe

Haluan kiittää Professori Kary Främlingia suostumisesta työn ohjaajaksi sekä valvojaksi.

Naapuriani Tarja Nissistä kiitän rakentavista kommenteista, maukkaista kaaliruuista ja takapihan pitämisestä siistinä.

Lisäksi haluan kiittää nykyistä työnantajaa joustavista työajoista sekä lomarahavapaaanomuksen hyväksymisestä vuonna 2017.

Helsinki, 27.11.2017

Mikko Rahikainen

# Sisällysluettelo

# Lyhenteet

| | |
|---|---|
| ACL | Agents Communication Language |
| B2MML | Business To Manufacturing Markup Language |
| CIM | Computer-integrated Manufacturing |
| DAI | Distributed Artificial Intelligence |
| DCOM | Distributed Component Object Model |
| ERP | Enterprise Resource Planning (suom. toiminnanohjausjärjestelmä) |
| FIPA | Foundation for Intelligent Physical Agents |
| HMI | Human Machine Interface |
| IoT | Internet of Things |
| ISA | International Society of Automation |
| JADE | Java Agent DEvelompment Framework |
| MAS | Multi Agent System (suom. moniagenttijärjestelmä) |
| MES | Manufacturing Execution System (suom. valmistuksenohjausjärjestelmä) |
| MESA | Manufacturing Enterprise Solutions Association |
| OPC | Open Platform Communications |
| PLC | Programmable Logic Controller |
| SCADA | Supervisory Control and Data Acquisition |
| SCM | Supply Chain Management |

# 1 Johdanto

## 1.1 Tausta

Industry 4.0 [21] on alun perin saksalainen termi, joka tarkoittaa neljättä teollista vallankumousta. Tavoitteena on kehitysaikojen lyhentäminen, tuotteiden massaräätälöinti, tuotannon joustavuus ja hajautus, sekä tuotannon tehokuus resurssien käytön suhteen. Teknologiapuolella tavoitteena on automatisointiasteen lisäys, digitalisaatio ja miniaturisaatio. Määrittäviä piirteitä ovat esimerkiksi pyrkimys älykkäisiin tehtaisiin, tuotannon itseorganisoituvuuteen, jakelu- ja hankintakanavien moninaistamiseen, sekä tuotteiden ja palveluiden räätälöintiin jopa yksittäisten tuotteiden osalta.

Jotta tuotannon tietojärjestelmät pystyisivät vastaamaan tulevaisuuden vaatimuksiin, voidaan olettaa, että niidenkin tulisi muuttua joustavammiksi ja itseorganisoituviksi. Teollisuusstandardit, kuten ISA-95 [2][4] määrittävät hierarkisen rakenteen jakamalla tuotannon tietojärjestelmät sekä niiden sisältämät toiminnot eri tasoille: laitetasolle, automaation logiikkatasolle, valmistuksenohjauksen sekä toiminnanohjauksen tasoille. Lisäksi on standardeja esim. panosautomaation tarpeisiin, kuten ISA-88 [1] tai laitteiden väliseen radiokommunikaatioon [17]. Nämä standardit eivät ota sellaisenaan kantaa älykkäiden tehtaiden tai itseorganisoituvien järjestelmien tarpeisiin.

Agenttipohjaisten järjestelmien ominaisuuksina voidaan pitää esim. toimivuutta seuraavankaltaisissa tilanteissa [30]:

- Toimivuus ympäristöissä, joissa ei voida käyttää keskitettyä arkkitehtuuria päätöksentekoon.
- Toimivuus hajautetuissa tai ajallisesti hajautetuissa ympäristöissä.
- Toimivuus ympäristöissä, joissa vaaditaan nopeaa vasteaikaa ja luotettavuutta.
- Agenttien käyttö simulaatiossa ja mallinnuksessa. Erityisesti, jos vaaditaan helppoa migraatiota simulaatiosta todelliseen ympäristöön.
- Monimutkaisten ongelmien ratkaisemisessa.
- Avointen järjestelmien suunnittelussa.

Nämä ominaisuudet vastaavat jo suoraan osaan Industry 4.0:n vaatimuksista, esim. joustavuuden ja hajautuksen osalta. Lisäksi simulointi on osa kyberfyysisiä järjestelmiä, joihin industry 4.0 tähtää. Luonnollinen kysymys on siis miten agenttipohjaiset ratkaisut voivat auttaa tuotannon tietojärjestelmien rakentamisessa neljännen teollisen vallankumouksen tarpeisiin.

Aihepiiri eli agenttien käyttö tuotannon tietojärjestelmissä on jo sinänsä varsin tutkittu aihe ja siitä löytyy runsaasti materiaalia. Tämän tutkimuksen motivaatio kumpuaa henkilökohtaisesta tiedonhalusta selvittää, mikä on agenttipohjaisten järjestelmien ja tutkimuksen nykytila tuotannon tietojärjestelmissä. Allekirjoittanut on toiminut yli 10 vuotta CGI Suomi Oy:n ja sen edeltäjien palveluksessa keskittyen

valmistuksenohjausjärjestelmien kehitykseen. Kuluneiden kymmenen vuoden aikana olen tutustunut mm. Siemensin ja Schneiderin MES-järjestelmiin sekä asiakkaalle räätälöityihin ratkaisuihin elintarvike- että terästeollisuudessa, mutta yksikään näistä järjestelmistä ei ole käyttänyt agenttipohjaisia ratkaisuja. Ennemmin voisi puhua monoliittisista ratkaisuista tai parhaimmillaan palvelupohjaisista ratkaisuista.

## 1.2 Tavoitteet

Työn tavoitteena on tutkia, miten MES-järjestelmän osia voidaan toteuttaa agenttipohjaisesti ja miettiä, miltä tulevaisuuden MES-järjestelmä voisi näyttää. Kysymyksiä, joihin tutkimuksella pyritään vastaamaan:

1. Mitä etuja agenttipohjaisesta toteutuksesta on?
2. Minkälaisia haasteita agenttipohjaisesta toteutuksesta aiheutuu?
3. Minkälainen voisi olla tulevaisuuden tuotannonohjausjärjestelmän arkkitehtuuri?

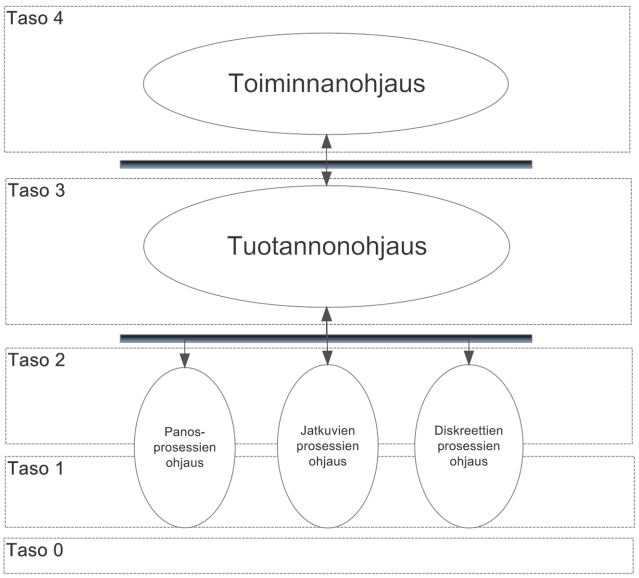
Työn tarkoituksena ei ole toteuttaa konkreettista tuotantojärjestelmää, vaan työn lopputulos on tämä tutkimusraportti ja kokeelliseen tutkimukseen kehitettävä järjestelmä.

## 1.3 Työn rakenne

Tämä opinnäytetyö rakentuu seuraavasti: kappaleessa 2 esitellään tuotannon tietojärjestelmien ja agenttipohjaisten menetelmien perusteet sekä teollisuuden tarpeet tulevaisuuden järjestelmille. Tässä kappaleessa myös referoidaan alan standardeja, kuten ISA-95[2], joiden määrittelemään MES-järjestelmien laajuuteen löydettyjä tutkimustuloksia tullaan vertaamaan. Kappaleessa 3 kuvataan tutkimuksessa käytetty tutkimusaineisto ja -menetelmät. Kappaleessa 4 esitellään kirjallisuustutkimuksen tulokset. Kappaleessa 5 kuvataan tutkimuksen tarpeisiin rakennetulla moniagenttijärjestelmällä saadut kokeelliset tulokset. Kappaleessa 6 vedetään tutkimuksen tulokset yhteen ja esitetään johtopäätökset. Lisäksi mietitään, mitkä kohteet voisivat olla potentiaalisia jatkotutkimuskohteita.

# 2 Taustatiedot

## 2.1 Tuotannon tietojärjestelmät




Kuva 1: Toiminnallinen hierarkia[2, s. 21]

ISA95 standardi määrittää toiminnallisen hierarkian tuotannon tietojärjestelmille. Kuvassa 1 sivulla 3 on yksinkertaistettu versio ISA95 standardin [2, s. 20] esittelemästä hierarkiasta:

Taso 0 käsittää fyysisen tuotantoprosessin.

Tasolla 1 ovat sensorit ja laitteet, joilla valvotaan ja manipuloidaan tuotantoprosessia.

Tasolla 2 on prosessinhallinnan aktiviteetit, joko manuaaliset tai automatisoidut, jotka pitävät prosessin stabiilina ja kontrollissa. Tason 2 järjestelmät käsittelevät tietoa tuntien, minuuttien, sekuntien ja sekunnin murto-osien tarkkuudella.

Tason 3 järjestelmät hallitsevat työnkulkua ja reseptejä haluttujen lopputuotteiden tuottamiseksi. Tällä tasolla pidetään kirjaa ja optimoidaan tuotantoprosessia. Tason 3 järjestelmien käsittelemä aikajänne on päiviä, vuoroja, tunteja, minuutteja ja sekunteja.

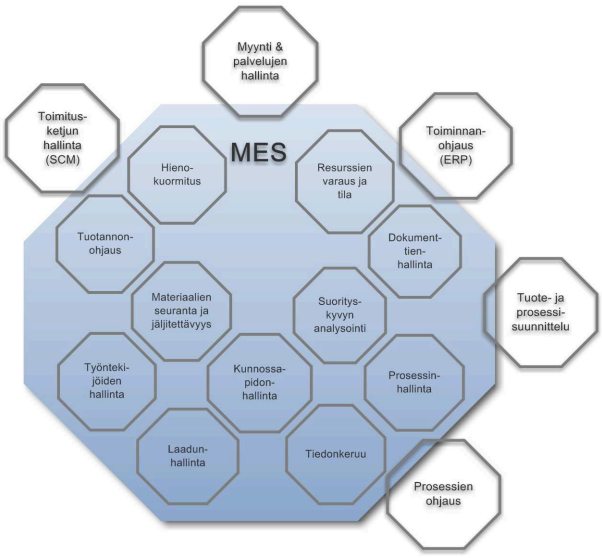
Tason 4 järjestelmät määrittelevät laitoksen tuotantoaikataulun, materiaalien käytön, tuotteiden lähettämisen ja kuljetuksen. Tällä tasolla myös pidetään kirjaa varastosaldoista. Tason 4 järjestelmien aikajänteet ovat kuukausia, viikkoja ja päiviä.

Eri tuotannon sovellukset voidaan sijoittaa standardin määrittämille tasoille seuraavasti:

- 2 - PLC ja SCADA järjestelmät.
- 3 - MES, PLM, CAD, CAM
- 4 - ERP, CRM, SCM, FIM, HRM, SRM

Rajat tasojen välillä eivät ole kiinteitä, vaan esim. MES-järjestelmä voi sisältää osin myös tasojen 2 tai 4 toiminnallisuuksia. [27, s. 9]

## 2.2 Valmistuksenohjausjärjestelmä




Kuva 2: MESA hunajakennomalli[34, s. 14]

MESA:n hunajakennomalli kuvassa 2 luetteloi valmistuksenohjausjärjestelmän tyypillisesti sisältämiä toiminnallisuuksia. Näitä ovat resurssien varaus ja status, hienokuormitus, tuotannonohjaus, dokumenttien hallinta, tiedonkeruu, työntekijöiden hallinta, laadunhallinta, prosessinhallinta, kunnossapidon hallinta, materiaalien seuranta ja jäljitettävyys sekä suorituskyvyn analysointi [34, s. 14]. ISA-95 standardi käyttää MESA:n määritelmää MES-järjestelmistä ja laajentaa niitä[4, s. 84].

## 2.3 Tuotannonsuunnittelu ja hienokuormitus

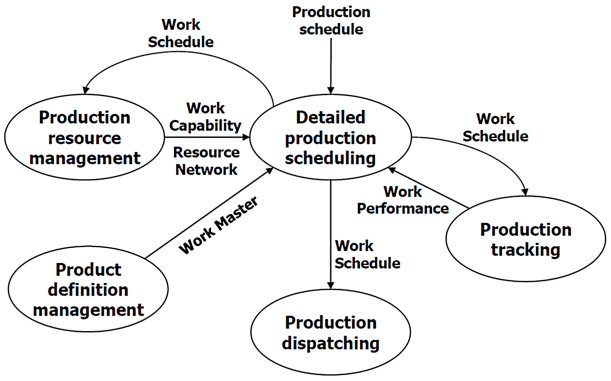
ISA-95 standardin määritysten mukaisesti tuotannonsuunnittelu on niiden toimintojen tai operaatioiden määritteleminen, joilla saavutetaan asetetut tavoitteet. Hienokuormitus on taas aktiviteetti, jolla toiminnot ja operaatiot kiinnitetään tietylle resurssille ja tietylle ajanhetkelle. Standardin mukaisessa hierarkiassa tuotannonsuunnittelu on hienokuormituksen yläpuolella, koska suunnittelu määrittää tavoitteet hienokuormitukselle. Suunnittelun lopputuloksena voi olla esimerkiksi tavoite tuottaa tietty määrä lopputuotetta kuukaudessa ja hienokuormituksen lopputuloksena työsuunnitelma "ajetaan tilausta X laitteella Y maanantaina klo 9"[4, s. 21].

Standardin mukaisesti työsuunnitelma resursseille muodostetaan tason 4 tuotantosuunnitelmasta (ERP). ISA-95 aktiviteettimallissa hienokuormitukseen voi liittyä seuraavia tehtäviä:

- Työsuunnitelman luominen ja ylläpito.

- Toteutuneiden tuotantomäärien vertaaminen suunniteltuun.
- Resurssien varatun kapasiteetin määrääminen.
- Tietojen keruu huolto-, laatu- ja varastonhallinnan operaatioista sekä
- mitä-jos -skenaarioiden simulointi.

Kuvassa 3 on kuvattuna hienokuormituksen liitännät muihin tuotannon operaatioihin ISA-95 standardin mukaisesti [4, s. 32].




Kuva 3: ISA-95 tuotannon hienokuormituksen aktiviteettimalli[4, s. 32]

## 2.4 Tuotannon järjestelmien integrointi

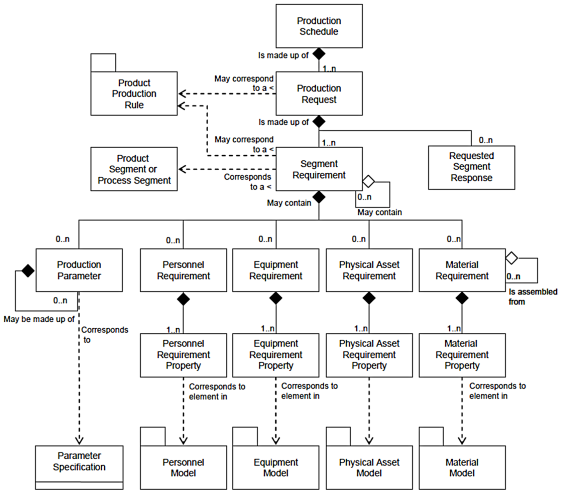
### 2.4.1 ISA-95

ISA-95 on standardi tuotantojärjestelmien integroimiseen, joka määrittelee rajapinnat tiedonvaihtoon yritystason aktiviteettien ja tuotanto- sekä ohjausjärjestelmätason aktiviteettien välille. Standardi keskittyy hierarkian (kuva 1) tasojen 3 ja 4 välisiin rajapintoihin. Standardin tavoitteena on vähentää rajapintojen implementointiin liittyviä riskejä, kustannuksia ja virheitä [2, s. 13]. Standardi jakautuu seuraaviin osiin:

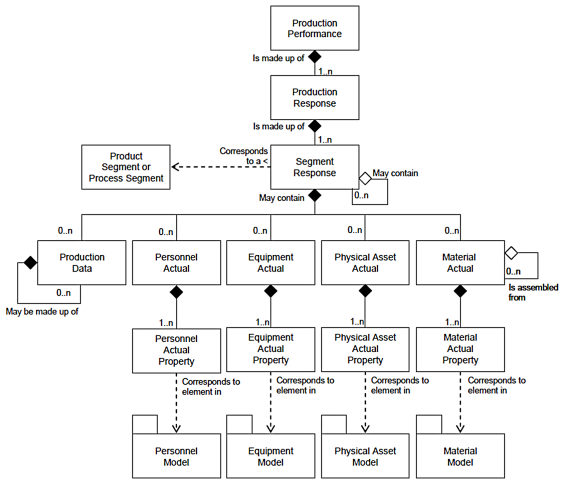
- Osa 1: Malli ja terminologia.
- Osa 2: Oliomalli systeemi-integraatioon.
- Osa 3: Aktiviteettimalli tuotannon operaatioiden hallinnasta.
- Osa 4: Oliomallit tuotanto-operaatioiden hallinnan aktiviteeteista.
- Osa 5: Liiketoiminnan ja tuotannon väliset transaktiot.
- Osa 6: Tuotannon operaatioiden transaktiot.

### 2.4.2 B2MML

B2MML (Business To Manufacturing Markup Language) on ISA-95 standardien XML-implementaatio, jota voidaan käyttää yrityksen tietojärjestelmien - kuten ERP tai CMS - integroimisessa tuotannon tietojärjestelmiin. ISA-95 standardi[2][3][4] määrittelee yhteiset käsitteet ja rakenteet, joilla eri järjestelmät kommunikoivat keskenään. Esimerkiksi kuvassa 4 on ISA-95 standardin mukainen kuvaus tuotantosuunnitelmasta (Production Schedule) ja kuvassa 5 tuotantojärjestelmän vastineesta (Production Performance).




Kuva 4: ISA-95: tuotantosuunnitelman oliomalli[3, s. 142]




Kuva 5: ISA-95: vastine tuotantojärjestelmästä[3, s. 144]

Liitteestä A löytyy sanitoitu esimerkki testijärjestelmästä otetusta B2MML-sanomasta, jossa on vapautettu tuotantotilaus järjestelmään (Production Schedule) sekä tilauksen valmistuttua ERP-järjestelmään lähetetty B2MML-muotoinen vastine (Production Performance).

### 2.4.3 OPC

OPC standardi on alun perin määritelty ratkaisemaan ongelma tiedonsiirrosta SCADA ja HMI järjestelmien ja prosessilaitteiden välillä. Ensimmäinen OPC standardi OPC DA on standardi tosiaikaisten arvojen siirtoon. Klassinen OPC on sidoksissa Microsoftin DCOM komponenttimalliin, mutta standardia on laajennettu kattamaan Web Servicet ja XML muotoinen data OPC XML DA ja OPC DX -standardeilla. Nykyisin OPC määrittelyt kattavat seuraavat standardit [27, ss. 136-138]:

- *OPC DA (Data Access)*: standardi reaaliaikaisten arvojen tiedonsiirtoon.
- *OPC A/E (Alarms and Events)*: standardi hälytysten ja tapahtumien siirtoon.
- *OPC HDA (Historical Data Access)*: standardi historiallisten arvojen siirtoon.
- *OPC DX (Data eXchange)*: standardi OPC-palvelinten väliseen kommunikaatioon.
- *OPC command*: standardi tilausten suorittamiseen.
- *OPC XML DA*: XML-pohjainen versio reaaliaikaisten arvojen välitykseen.
- *OPC UA*: alustariippumaton versio OPC-standardeista ilman DCOM-teknologiaa, joka yhdistää kaikki aiemmat standardit saman spesifikaation alle.

ISA-95 hierarkiassa (kuva 1) OPC voi siis toimia MES-kontekstissa rajapintana tason 3 ja alempien tasojen väillä.

## 2.5 Industry 4.0 ja älykkäät tehtaat

Visio tulevaisuuden tuotannosta on modulaariset ja tehokkaat tuotantojärjestelmät, joissa tuotteet ohjaavat omaa tuotantoprosessiaan. Näin on tarkoitus toteuttaa yksilöllisten tuotteiden tuotanto yhden tuotteen eräkoolla, ilman että tarvitsee luopua massatuotannon taloudellisista eduista. Termi "Industry 4.0"kuvaa tätä suunniteltua neljättä teollista vallankumousta. [21]

Industry 4.0:n sosiaalisia, taloudellisia ja poliittisia vaikuttimia ovat: [21]

- *Lyhyet kehitysajat*: kehitys- ja innovointiaikoja pitää lyhentää. Korkea innovointiaste on tulossa ratkaisevaksi menestystekijäksi yrityksille.
- *Yksilölliset tuotteet*: markkinoiden muutos myyjän markkinoista ostajan markkinoiksi on ollut selvä kehityssuunta jo vuosikymmeniä. Tämä tarkoittaa, että ostaja voi määritellä kaupan ehdot ja johtaa enenevissä määrin tuotteiden kasvavaan kustomointiin. Ääritapauksissa puhutaan yksilöllisistä tuotteista eli yhden kappaleen eräkoosta.
- *Joustavuus*: Uusien vaatimusten takia joustavuus tuotekehityksessä, erityisesti tuotannossa, on tarpeellista.

- *Hajautettavuus*: erilaisten yksilöllisten ehtojen takia tarvitaan nopeampaa päätöksentekoa. Siksi organisaatioiden hierarkioita pitää madaltaa.

- *Resurssienkäytön tehokkuus*: resurssien niukkuus ja niukkuuteen liittyvät hintapaineet sekä ekologiset kestävään kehitykseen teollisuudessa liittyvät tekijät vaativat resurssienkäytön tehokkuutta. Tavoite on taloudellinen ja ekologinen tehokkuuden lisäys.

Teollisesta kontekstista syntyviä vaikutteita taas ovat: [21]

- *Mekanisaation ja automatisaation lisääminen*: tuotantoprosessissa käytetään enemmän ja enemmän teknisiä apuvälineitä. Lisäksi automaatioratkaisut laajenevat autonomisiksi tuotantosoluiksi, jotka itsenäisesti ohjaavat ja optimoivat tuotantoa sen eri vaiheissa.

- *Digitalisaatio ja tietoverkot*: tuotannon lisääntyvä digitalisaatio tuottaa entistä enemmän dataa, jota voidaan käyttää ohjaukseen ja analysointiin. Digitaaliset prosessit kehittyvät rinta rinnan teknisten ratkaisujen ja digitaalisten palvelujen kanssa. Tämä johtaa lopulta täysin digitalisoituihin ympäristöihin.

- *Miniaturisointi*: tietoteknisten laitteiden koon pieneneminen mahdollistaa uusia sovelluksia, erityisesti tuotannossa ja logistiikassa.

Älykäs tehdas on tuotannon kyberfyysinen (engl. cyber-physical) järjestelmä, joka integroi fyysiset entiteetit kuten laitteet, kuljettimet ja tuotteet informaatiojärjestelmiin, kuten MES- ja ERP-järjestelmät, joustavaa ja ketterää tuotantoa varten. [36]

Pääasialliset vaatimukset ja haasteet järjestelmille älykkäiden tehtaiden toteutuksessa ovat [9, p. 73]:

- *Hyvin joustava ja uudelleen konfiguroitava:* Markkinat asettavat kovia vaatimuksia valmistavalle teollisuudelle. Jotta yritys voisi pysyä kilpailukykyisenä, sen pitää pystyä vastaamaan asiakkaan vaatimuksiin nopeasti. Tuotantoprosesseja pitää pystyä muuttamaan ilman, että vaaditaan pitkiä tuotantoseisakkeja. Joustavuus on hyvin tärkeää, sillä tuotteiden räätälöinti on vakava haaste tulevaisuuden tehtaille. Jatkossa massatuotanto ei enää toimi kustannusten pienentämisessä kuten nykyään. Tulevaisuuden tehtaan koko on pienempi ja tehdas voidaan sijoittaa pienempiin laitoksiin.

- *Modulaarisuus uudelleenkäytettävillä komponenteilla:* Jäykkiä keskitettyjä tuotantojärjestelmiä tulee välttää. Kun älykkään tehtaan kapasiteetti on joustava, tuotantolinjat koostuvat löyhästi kytketyistä komponenteista. Näitä komponentteja on tarkoitus käyttää myös muiden tuotteiden valmistuksessa, kuin niiden, joita alun perin on suunniteltu.

- *Liitetty ylempiin yrityksen hallinnon tasoihin:* Nykyään tehtaita ei ole välttämättä kytketty yrityksen strategiseen tasoon. Usein insinöörit hoitavat manuaalisesti valmistuserien tuottamisen tuotantotilausten perusteella. Älykäs tehdas

kytketään yrityksen keskeisiin liiketoimintaprosesseihin ja toimii reaaliaikaisen tiedon pohjalta: esim. uusien asiakkaiden tai liiketoimintamahdollisuuksien ilmaantuminen voi vaikuttaa tuotannon päätöksiin.

- *Tarkka jäljitettävyys tuotantoketjussa:* Tulevaisuuden tehdas mahdollistaa materiaalien ja tuotettujen tuotteiden jäljitettävyyden. Jotkut teollisuuden alat, kuten elintarviketuotanto, vaativat täyden tuotteen elinkaaren jäljitettävyyden hankintaketjusta lopputuotteisiin ja mahdollisesti viranomaisten vaatiman sertifioinnin. Tästä aiheutuu haasteita nykyisille tekniikoille mallinnuksessa, analysoinnissa ja jäljitettävyystiedon hallinnassa.

- *Ihmislähtöinen:* Tulevaisuuden tehdas sisältää teknologioita, jotka on erityisesti tarkoitettu työntekijöille luotettavuuden parantamiseksi.

- *Standardeihin pohjautuva:* Jotta kommunikaatio eri sidosryhmien välillä olisi helppoa tuotantojärjestelmissä, avoimista standardeista tulee käytännössä pakollisia. Näistä standardeista BPMN on käytössä prosessien mallinnuksessa ja ISA-95 standardi määrittelee yhteisen sanaston ja rakenteen tuotannon järjestelmien suunnitteluun.

- *Semanttisesti erotettu:* Suunnitteluprosessin tulee erottaa erilaiset tarpeet, kuten mekaaninen, sähköinen ja säätöjärjestelmien suunnittelu omiksi domainspesifisiksi malleikseen. Haasteeksi tulee erillisten mallien yhdistäminen ja orkestointi, jotta voidaan simuloida ajettavia prosesseja ja verifioida suunnitelmien toteutuskelpoisuus.

Artikkelissa "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination"[36] älykkään tehtaan arkkitehtuuriksi ehdotetaan mallia, jossa lattiatason laitteet, tuotteet, kuljettimet jne. on yhdistetty itseorganisoituvaksi koneiden verkoksi. Agentit, jotka edustavat entiteettejä, muodostavat moniagenttijärjestelmän (engl. Multi Agent System, MAS) ja neuvottelevat tuotantoprosessin etenemisestä. Agentit ovat teollisen internetin välityksellä liittettynä pilveen, jonne kerätään data analyysiä ja ohjausta varten.

## 2.6 Agentti

Woolridge ja Jennings [38, s. 116] määrittelevät agentin joko laite- tai sovelluspohjaiseksi järjestelmäksi, jolla on seuraavat ominaisuudet:

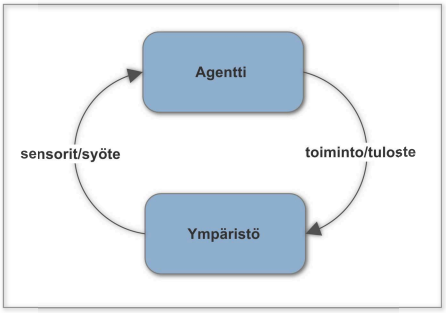
- *Autonomia* - Agentit toimivat ilman ulkopuolisten tahojen suoraa ohjausta. Agentit hallitsevat omia toimintojaan ja sisäistä tilaansa.

- *Sosiaaliset kyvyt* - Agenteilla on kyky toimia vuorovaikutuksessa toisten agenttien - tai mahdollisesti - ihmisten kanssa jonkinlaisen agenttien kommunikointiin tarkoitetun kielen välityksellä.

- *Reaktiivisuus* - Agentit havainnoivat ympäristöään ja reagoivat siinä tapahtuviin muutoksiin. Ympäristö voi olla esim. fyysinen maailma, käyttöliittymä, joukko muita agentteja, internet tai joku näiden yhdistelmä.

- *Proaktiivisuus* - Agentit eivät vain reagoi ympäristöönsä, vaan agentit pystyvät myös toimimaan aloitteellisesti päämääriensä saavuttamiseksi.

Yksinkertainen tapa agentin havainnollistamiseksi on sovellusprosessi (esim. UNIX taustaprosessi), jolla on edellä mainitut ominaisuudet. Esimerkiksi tietojenkäsittelytieteessä agentti on itsenäinen rinnakkaisesti ajettava prosessi, joka kapsuloi oman tilansa ja kommunikoi muiden agenttien kanssa viestejä välittämällä. Tämä voidaan nähdä luonnollisena kehityksenä oliopohjaiselle rinnakkaisohjelmoinnille [38, s. 117].




Kuva 6: Agentti ympäristössään - agentti saa syötteen sensoreiden kautta ympäristöstä ja tuottaa tulosteena toimintoja, jotka vaikuttavat ympäristöön. Interaktio on yleensä jatkuvaa eikä pääty. [37, s. 29]

Agentin toimintaympäristön ominaisuuksia suhteessa siinä toimivaan agenttiin voidaan määritellä esim. seuraavasti [37, s. 82]:

- *Tiedettävyys (engl. knowable)* eli missä määrin ympäristön tila on agentin tiedossa.
- *Ennustettavuus (engl. predictable)* eli missä määrin agentti voi ennustaa ympäristön toimintaa.
- *Kontrolloitavuus (engl. controllable)* eli missä määrin agentti voi muokata ympäristöä.
- *Historiallisuus (engl. historical)* eli riippuvatko tulevaisuuden tilat koko historiasta vai vain nykyisestä tilasta.
- *Tarkoitusperäisyys (engl. teleological)* eli ovatko ympäristön osat tarkoituksenmukaisia esim. onko olemassa muita agentteja.
- *Tosiaikaisuus (engl. real-time)* eli voiko ympäristö muuttua agentin miettiessä.

### 2.6.1 Holoni

Holoni on termi, joka on johdettu kreikkalaisesta sanasta *holos* “kokonaisuus” ja suffiksista *on* “osa”. Termi on alunperin Arthur Kostrelin muodostama kirjassaan *The Ghost in the Machine* ja sitä käytetään kuvaamaan toistuvien rakenteiden sisäkkäisiä

hierarkioita. Holoni on yleensä määritelty itsesimilaarisena rakenteena, joka koostuu muista holoneista [10, s. 2].

Kaksi holonin tärkeää ominaisuutta ovat [22, s. 983]:

- *Autonomia*: holonien vakaus syntyy kyvystä toimia autonomisesti ennalta-arvaamattomissa olosuhteissa.
- *Yhteistyö (engl. co-operation)*: holoneilla on kyky tehdä yhteistyötä ja näin muodostaa suurempia kokonaisuuksia.

Holoni voi esittää fyysistä tai loogista aktiviteettia, kuten robottia, laitetta, tilausta, joustavaa tuotantojärjestelmää tai vaikka ihmistä. Holonilla on tieto itsestään ja ympäristöstään. Esimerkiksi holoni sisältää informaation prosessointiosuuden ja fyysisen prosessointiosuuden, mikäli holoni vastaa fyysistä laitetta, kuten teollista robottia. [22, s. 983]

Agenttipohjaisen ja holonisen tuotannon paradigmat on kehitetty samojen perusperiaatteiden mukaan: autonomia, yhteistyö, entiteettien ja toimintojen hajautus.

Agenttiteknologian ja holonien välisistä eroista: [22, s. 983]

- Alunperin agentit on kehitetty tietämystekniikan ja holonit tietokoneintegroidun tuotannon (CIM) puitteissa.
- Holoni on konsepti ja agentti on sekä konsepti että teknologiaa. Holonin käsite tai holoninen tuotantojärjestelmä voidaan implementoida käyttäen agenttiteknologiaa.
- Agenttiin verrattuna holoni voi koostua useammasta alitason holonista.
- Holoni-käsite tukee integraatiota fyysisiin laitteisiin, koska holoni koostuu sekä loogisista että fyysisistä komponenteista.
- Agentti ei voi taata reaaliaikaisia vaatimuksia, mutta holonin täytyy pystyä takaamaan reaaliaikavaatimukset luotettavan toiminnan varmistamiseksi.

Holoni voidaan nähdä reaktiivisena agenttina, joka toimii kovien reaaliaikavaatimusten kanssa ja jolla on yhteys fyysisiin laitteisiin.

## 2.7 Moniagenttijärjestelmät

Moniagenttijärjestelmät (engl. Multi-Agent Systems, MAS) ovat hajautettuja järjestelmiä, jotka koostuvat autonomisista ohjelmallisista toimijoista eli agenteista [32].

Moniagenttiympäristöjen tyypillisiä piirteitä [37, s. 81]:

- Moniagenttiympäristöt tarjoavat infrastruktuurin määrittelemällä kommunikaatio- ja interaktioprotokollat.
- Moniagenttiympäristöt ovat tyypillisesti avoimia eikä niillä ole yksittäistä suunnittelijaa.

- Moniagenttiympäristöt sisältävät agentteja, jotka ovat autonomisia ja hajautettuja ja jotka voivat olla joko oman edun tavoittelijoita tai toimia yhteistyössä.

Weiss [37, s. 82] luettelee lisäksi moniagenttiympäristöjen piirteitä sekä ympäristön ja agentin välisen vuorovaikutuksen piirteitä, jotka on koottu taulukkoon 1.

| Ominaisuus | Mahdolliset arvot |
|---|---|
| Suunnittelun autonomia | alusta / interaktioprotokolla / kieli / sisäinen arkkitehtuuri |
| Kommunikaatioinfrastruktuuri | Jaettu muisti (engl. blackboard) tai viestipohjainen;<br>Jatkuva yhteys tai yhteydetön;<br>Point-to-point, multicast tai broadcast;<br>push/pull;<br>synkroninen tai asynkroninen |
| Hakemistopalvelu | Valkoiset-/keltaiset sivut<br>(engl. white pages, yellow pages) |
| Viestintä protokolla | KQML, HTTP & HTML, OLE, CORBA, DCOM |
| Välityspalvelut | ontologiapohjaiset/transaktiot |
| Tietoturvapalvelut | aikaleimat/autentikointi |
| Maksupalvelut | laskutus/valuutta |
| Operaatioiden tuki | arkistointi/redundanssi/palautus/kirjanpito |

Taulukko 1: Moniagenttiympäristöjen ominaisuuksia [37, s. 82]

### 2.7.1 FIPA-standardit

FIPA-standardien tavoite on edistää keskenään yhteensopivien agenttipohjaisten sovellusten ja järjestelmien tekemistä. Standardit kattavat määrittelyjä agenttipohjaiselle infrastruktuurille ja sovelluksille. Infrastruktuurimäärittelyt sisältävät kielen agenttien väliseen kommunikointiin, agenttipalvelut ja näitä tukevat hallinto-ontologiat. Lisäksi standardit määrittelevät joukon sovellusalueita, kuten henkilökohtainen matkustusassistentti ja verkkojen hallinta ja provisiointi [11, p. 3].

FIPA-standardien keskiössä on agenttien välinen kommunikointi, missä agentin voivat välittää toisilleen semanttisesti tarkoituksenmukaisia viestejä sovelluksen vaatimien tehtävien suorittamiseksi.

FIPA määrittelee abstraktin arkkitehtuurin, jossa agentit voivat vaihtaa viestejä käyttäen erilaisia viestinvälitystapoja, erilaisia agenttien väliseen kommunikointiin tarkoitettuja kieliä (engl. Agent Communication Languages, ACL) tai erilaisia sisältökieliä (engl. content languages). FIPA-arkkitehtuuri kattaa seuraavia osa-alueita[11, p. 3]:

- Malli palveluista ja palveluiden löytämisestä agenteille ja muille palveluille.
- Viestinvälityksen yhteensopivuus.
- Tuki erilaisille ACL esitysmuodoille.
- Tuki erilaisille sisältökielille.

- Tuki useille hakemistopalveluille.

FIPA-kommunikaatioaktikirjasto [12] määrittelee joukon standardeja kommunikaatioakteja (engl. communicative act). Kirjaston tavoitteena on taata yhteensopivuus ja mahdollistaa uudelleenkäyttö sekä tarjota toiminnot ja nimikkeen FIPA ACL-kielen käyttöön. Taulukossa 2 on listattuna kirjastossa määritellyt kommunikaatioaktit sekä jokaisesta lyhyt kuvaus.

Kommunikaatioaktien lisäksi FIPA-spesifikaatiot määrittelevät myös protokollia agenttien väliseen kommunikointiin, kuten esimerkiksi Contract Net -protokollan, jossa aloittajan roolissa oleva agentti pyytää muilta agenteilta tarjousta tietyn toimenpiteen suorittamisesta. Aloittaja valitsee lopullisen työn suorittajan tarjousten perusteella käyttäen perusteena esimerkiksi työn suorituksen kustannusta tai suoritusaikaa [13].

| Kommunikaatioakti | Kuvaus |
|---|---|
| Accept Proposal | Hyväksyy aiemmin lähetetyn tarjouksen toiminnan suorittamista. |
| Agree | Hyväksyntä jonkun toiminnon suorittamisesta, mahdollisesti tulevaisuudessa. |
| Cancel | Agentti tiedottaa toiselle agentille, ettei sillä ole enää tarvetta toisen agentin suorittamalle toimenpiteelle. |
| Call for Proposal | Tarjouksien pyyntö toiminnon suorittamisesta. |
| Confirm | Lähettäjä ilmoittaa vastaanottajalle, että propositio on tosi, missä vastaanottaja on tiedetty epätietoiseksi proposition totuusarvosta. |
| Disconfirm | Lähettäjä ilmoittaa vastaanottajalle, että propositio on epätosi, missä vastaanottajan oletetaan uskovan proposition olevan tosi. |
| Failure | Kerrotaan toiselle agentille, että toimintoa yritettiin, mutta se epäonnistui. |
| Inform | Lähettäjä tiedottaa vastaanottajalle, että propositio on tosi. |
| Inform If | Makrotoiminto tiedon lähettämiseksi vastaanottajalle onko propositio tosi vai ei. |
| Inform Ref | Makrotoiminto, jolla lähettäjä kertoo vastaanottajalle entiteetin kuvauksen (viite). |
| Not Understood | Lähettäjä kertoo vastaanottajalle, ettei ymmärtänyt vastaanottajan suorittamaa toimintoa. Esim. vastaanottajan lähettämää viestiä. |
| Propagate | Lähettäjä pyytää vastaanottajaa lähettämään viestin edelleen. |
| Propose | Tarjouksen lähettäminen jonkin toiminnon suorittamisesta tietyillä esiehdoilla. |
| Proxy | Lähettäjä haluaa vastaanottajan valitsevat kohdeagentit kuvauksen perusteella ja lähettävän sisältönä olevan viestin kohdeagenteille. |
| Query If | Lähettäjä kysyy vastaanottajalta proposition totuusarvoa. |
| Query Ref | Lähettäjä kysyy vastaanottajalta entiteettiä johon viitelause viittaa. |
| Refuse | Kieltäytyminen toiminnon suorittamisesta ja kieltäytymisen syyn selitys. |
| Reject Proposal | Tarjouksen hylkääminen. |
| Request | Pyyntö vastaanottajalle suorittaa jokin toimenpide. |
| Request When | Pyyntö vastaanottajalle suorittaa toimenpide, kun määrätyn proposition totuusarvoksi vaihtuu tosi. |
| Request Whenever | Pyyntö vastaanottajalle suorittaa toimenpide, kun määrätyn proposition totuusarvoksi vaihtuu tosi (joka kerta). |
| Subscribe | Pyytää vastaanottajaa ilmoittamaan, kun entiteetin tilassa tapahtuu muutoksia. |

Taulukko 2: FIPA kommunikaatioaktit [12]

## 2.8 Laskennan teoreettisesta kompleksisuudesta

Garey ja Johnson ovat kirjassa *Computers and Intractability* [14, ss. 241 - 243] listanneet NP-täydellisiksi ongelmiksi mm. seuraavat päätösongelmat, jotka ovat tuotannonohjausjärjestelmien kannalta mielenkiintoisia:

- Open-Shop hienokuormitusongelma (Open-Shop Scheduling)
- Flow-Shop hienokuormitusongelma (Flow-Shop Scheduling)
- Job-Shop hienokuormitusongelma (Job-Shop Scheduling)
- Tuotannonsuunnittelu (Production Planning)
- Henkilöstön aikataulutus (Staff Scheduling)

Kuten kirjassa todetaan [14, s. 14]: *Tieto, että ongelma on NP-täydellinen, vähintäänkin vihjaa, ettei ilman suurta läpimurtoa ongelma ole ratkaistavissa polynomisessa*

*ajassa.* Tiedon merkitystä havainnollistaa hyvin seuraava taulukko 3, jossa on vertailtu polynomisia ja eksponentiaalisia suoritusaikoja suhteessa ongelman kokoon.

| Suoritus-ajan-funktio | Koko $n$ | | | | | |
|---|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 | 60 |
| $n$ | 0,00001 s | 0,00002 s | 0,00003 s | 0,00004 s | 0,00005 s | 0,00006 s |
| $n^2$ | 0,0001 s | 0,0004 s | 0,0009s | 0,0016 s | 0,0025 s | 0,0036 s |
| $n^3$ | 0,001 s | 0,008 s | 0,027s | 0,064 s | 0,125 s | 0,216 s |
| $n^5$ | 0,1 s | 3,2 s | 24,3 s | 1,7 min | 0,125 s | 0,216 s |
| $2^n$ | 0,001 s | 1,0 s | 17,9 min | 12,7 vrk | 35,7 v | 366 vuosisataa |
| $3^n$ | 0,059 s | 58 min | 6,5 v | 3855 vuosisataa | $2 * 10^8$ vuosisataa | $1,3 * 10^{13}$ vuosisataa |

Taulukko 3: Muutamien polynomisten ja eksponentiaalisten suoritusaikojen vertailu [14, s. 7]

# 3 Tutkimusaineisto ja -menetelmät

## 3.1 Aineisto

Taustatiedot ja edeltävä tutkimus on esitelty kappaleessa 2. Taustatiedot sekä tutkimuksen aineisto on kerätty alan tieteellisistä artikkeleista, konferenssijulkaisuista, oppikirjoista sekä teollisuusstandardeista. Aineisto on koottu käyttäen Aalto-yliopiston kirjaston tiedonhakujärjestelmiä.

Aihepiiristä on jo tehty aiemmin koostetyyppisiä tutkimuksia [35][28], joita käytetään työn pohjana. Näiden lisäksi on pyritty etsimään uusinta aineistoa sekä alkuperäisiä lähteitä.

## 3.2 Menetelmät

Kappaleessa 2.2 esiteltyä ISA-95 ja MESA-mallin mukaista MES-järjestelmien toteuttamia toiminnallisuuksia verrataan tutkimuksessa aineistosta löydettyihin esimerkkeihin. Työssä keskitytään yhteen MES-järjestelmien osa-alueeseen eli hienokuormitukseen.

Kappaleessa 4 on koottu alan lähteistä hienokuormitukseen liittyviä tutkimustuloksia. Kokeellisessa osuudessa kappaleessa 5 rakennetaan agenttipohjainen hienokuormitus käyttäen markkinamekanismia sekä geneettistä algoritmia, sekä vertaillaan algoritmien suorituskykyä simuloiduilla ongelmilla.

Kirjallisuus- ja kokeellisella tutkimuksella pyritään vastaamaan kappaleessa 1.2 esitettyihin kysymyksiin:

1. Mitä etuja agenttipohjaisesta toteutuksesta on?
2. Minkälaisia haasteita agenttipohjaisesta toteutuksesta aiheutuu?
3. Minkälainen voisi olla tulevaisuuden tuotannonohjausjärjestelmän arkkitehtuuri?

Valittu MES-näkökulma erottaa tämän tutkimuksen aiemmista tutkimuksista, kuten edellä mainituista koostetyyppisistä tutkimuksista.

# 4 Tuotannon hienokuormitus

## 4.1 Yleinen määrittely

Kaikissa hienokuormitusongelmissa töiden ja koneiden määrä on rajallinen. Töiden (jobs) määrää on $n$ ja koneiden määrä $m$. Yleensä alaviite $j$ viittaa työhön ja alaviite $i$ viittaa koneeseen. Jos työhön kuuluu monta operaatiota $o$, pari $(i, j)$ viittaa työn $j$ operaatioon koneella $i$. Työhön $j$ liittyvät seuraavat tiedot: [31]

- Suoritusaika ($p_{ij}$): $p_{ij}$ on työn $j$ suoritusaika koneella $i$. Alaviitettä $i$ ei käytetä, jos työn suoritusaika ei riipu suorittavasta koneesta tai suorittavia koneita on vain yksi.
- Vapautusaika (engl. Release date) ($r_j$) on aika, jolloin työ on vapautettu tuotantoon eli aikaisin ajankohta, jolloin työtä $j$ voidaan alkaa suorittaa.
- Valmistumisaika (engl. Due date) ($d_j$) on työn $j$ luvattu toimitus- tai valmistumisaika. Työn valmistuminen tämän jälkeen on sallittua, mutta ajan ylittämiseen voi liittyä sakko. Mikäli luvattua valmistumisaikaa ei saa ylittää, on kyseessä *deadline*, jota merkitään symbolilla $\bar{d}_j$.

## 4.2 Erilaisten varianttien nimeäminen

Hienokuormitusongelmia kuvataan kolmikolla $\alpha|\beta|\gamma$, jossa $\alpha$ kuvaa tuotantolaitoksen konfiguraation, $\beta$ joukon rajoitteita ja $\gamma$ tavoitefunktion.

Parametri $\alpha$ kuvaa tuotantolaitoksen rakenteen, vaiheiden määrän sekä koneiden määrän ja ominaisuudet vaiheittain. Parametri $\alpha$ jakautuu neljään aliparametriin: $\alpha_1$, $\alpha_2$, $\alpha_3$ ja $\alpha_4$. Näistä $a_1$ ilmaisee konfiguraation, esim. jos kyseessä on *Hybrid Flow Shop* -ongelma on $\alpha_1 = FH$. Aliparametri $\alpha_2$ on tuotantovaiheiden määrä. Aliparametrit $\alpha_3$ ja ja $\alpha_4$ kuvaavat vaiheen koneiden tyypin. Notaatio $(\alpha_3\alpha_4)^k$ tarkoittaa, että vaiheessa $k$ on $\alpha_4$ rinnakkaista konetta tyyppiä $\alpha_3$. Aliparametri $\alpha_3 \in \varnothing, P, Q, R$, missä $P$ tarkoittaa identtisiä rinnakkaisia koneita, $Q$ rinnakkaisia eri nopeuksisia koneita ja $R$ erilaisia rinnakkaisia koneita. Jos vaiheessa on vain yksi kone, $\alpha_3 = \varnothing$ [33].

Parametri $\beta$ listaa ne rajoitteet ja oletukset, jotka eivät sisälly standardiin ongelmaan. Yleisimpiä rajoitteita ja oletuksia ovat[33]:

- $r_j$ tarkoittaa, ettei työtä $j$ voida aloittaa suorittamaan ennen sen vapautusta tuotantoon ajankohtana $r_j$
- *prmu* indikoi, että työt suoritetaan jokaisessa vaiheessa samassa järjestyksessä.
- *prec* tarkoittaa, että eri töiden koneella suoritettavilla operaatioilla on järjestysrajoitteita.
- $M_j$ tarkoittaa, että työ $j$ voidaan prosessoida joukolla laitteita $M_j$ vaiheessa $k$.
- $S_{sd}$ tarkoittaa, että asetusajat riippuvat operaatioiden järjestyksestä.

- *prmp* merkitsee, että työt voidaan tarvittaessa keskeyttää.
- *block* merkitsee, että puskurit vaiheiden välillä ovat rajoitetut. Töiden täytyy odottaa edellisessä vaiheessa kunnes puskureissa on tilaa.
- *recr* merkitsee, että työt voivat käydä samassa prosessivaiheessa useasti.
- *unavail* merkitsee, että koneet eivät ole käytettävissä kokoaikaisesti.
- $no - wait$ merkitsee, että työt eivät voi odottaa peräkkäisten vaiheiden välillä. Tällöin tuotantolaitos operoi *ensimmäinen sisään, ensimmäinen ulos* (FIFO) periaatteella.
- $p_j = p$ tarkoittaa, että kaikki suoritusajat ovat $p$.
- $size_{jk}$ tarkoittaa, että operaatio $o_{jk}$ täytyy suorittaa $size_{jk}$ laitteella yhtäaikaisesti.

Taulukossa 4 on yleisiä optimoitavia tavoitefunktioita hienokuormitukselle. Jos $C_j$ on työn $j$ valmistumisaika , niin läpimenoaika työlle $F_j$ on aika jonka työ viettää järjestelmässä eli $F_j = C_j - r_j$. Viive on tällöin $L_j = C_j - d_j$, myöhästyminen $T_j = max\{C_j - d_j, 0\}$ ja etuaikaisuus $E_j = max\{d_j - C_j, 0\}$ [33].

| Notaatio | Kuvaus | Tarkoitus |
|---|---|---|
| $C_{max}$ | $max_j C_j$ | Myöhäisin valmistumisaika |
| $F_{max}$ | $max_j(C_j - r_j)$ | Suurin läpimenoaika (engl. flow time) |
| $L_{max}$ | $max_j(L_j)$ | Suurin viive (engl. lateness) |
| $T_{max}$ | $max_j(T_j)$ | Suurin myöhästyminen (engl. tardiness) |
| $E_{max}$ | $max_j(E_j)$ | Suurin etuaikaisuus (engl. earliness) |
| $\bar{C}$ | $\sum C_j$ | Keskimääräinen valmistumisaika/summa |
| $\bar{C}^w$ | $\sum w_j C_j$ | Keskimääräinen painotettu valmistumisaika/summa |
| $\bar{F}$ | $\sum F_j$ | Keskimääräinen läpimenoaika/summa |
| $\bar{F}^w$ | $\sum w_j F_j$ | Keskimääräinen painotettu läpimenoaika/summa |
| $\bar{T}$ | $\sum T_j$ | Keskimääräinen myöhästyminen/summa |
| $\bar{T}^w$ | $\sum w_j T_j$ | Keskimääräinen painotettu myöhästyminen/summa |
| $\bar{U}$ | $\sum U_j$ | Myöhästyneiden töiden määrä |
| $\bar{U}^w$ | $\sum w_j U_j$ | Painotettu myöhästyneiden töiden määrä |
| $\bar{E}$ | $\sum E_j$ | Keskimääräinen etuaikaisuus/summa |
| $\bar{E}^w$ | $\sum w_j E_j$ | Keskimääräinen painotettu etuaikaisuus/summa |

Taulukko 4: Yleisiä optimoitavia tavoitefunktioita [33]

## 4.3 Erilaiset hienokuormitusongelmat

### 4.3.1 Flow Shop -ongelma, $\alpha_1 = Fm$

Koneita on $m$ kappaletta ja kaikki koneet ovat peräkkäin. Jokainen työ prosessoidaan jokaisella $m$:llä koneella ja kaikkien töiden tulee seurata samaa reittiä, esim. työ pitää ensin prosessoida koneella 1 ja sen jälkeen koneella 2. Kun työ on prosessoitu yhdellä koneella, se siirtyy jonoon seuraavalle koneelle. Yleensä kaikki jonot toimivat "ensimmäinen sisään, ensimmäinen ulos" (FIFO) -periaatteella [31].

### 4.3.2 Hybrid Flow Shop -ongelma $\alpha_1 = FHm$ tai Flexible flow shop -ongelma $\alpha_1 = FFm$

Hybrid Flow Shop (HFS)[33] ongelmat ovat yleisiä tuotantoympäristöissä, joissa suoritettavien töiden määrä on $n$:n ja jokaiselle työlle on $m$ prosessivaihetta. Ongelmasta on useampia variantteja, joilla on seuraavat yhteiset piirteet:

- Prosessivaiheiden määrä $m$ on vähintään 2
- Jokaiselle vaiheella $k$ on käytössä rinnakkaisia koneita $M^{(k)} >= 1$ kappaletta, ja ainakin yhdelle vaiheelle $M^{(k)} > 1$
- Kaikki työt käyvät läpi järjestyksessä samat prosessivaiheet: vaihe 1, vaihe 2, ... , vaihe $m$. Työ voi ohittaa prosessivaiheita, kunhan se on prosessoitu ainakin yhdessä niissä.

### 4.3.3 Job Shop -ongelma, $\alpha_1 = Jm$

Job shop -ongelmassa $m$:llä koneella, jokaisella työllä on oma ennalta määrätty reitti. Variaatioita ovat mm. käykö työ jokaisella koneella kerran vai voiko työ käydä jollain koneella useasti [31].

### 4.3.4 Flexible job shop -ongelma, $\alpha_1 = FJc$

Flexible job shop on yleistys job shop -ongelmasta rinnakkaiskoneympäristöön. Yksittäisten koneiden sijasta on olemassa $c$ työpistettä (engl. work center), joissa jokaisessa on tietty määrä rinnakkaisia koneita. Jokaisella työllä on sen oma reitti ja jokaisessa työpisteessä yksittäinen työ vaatii prosessointia ainoastaan yhdeltä koneelta ja mikä tahansa mahdollisista koneista kelpaa. Kuten job shop-ongelmassa, yksi variaatio on käykö työ jokaisella työpisteellä kerran vai voiko työ käydä jollain työpisteellä useasti [31].

### 4.3.5 Open Shop -ongelma, $\alpha_1 = Om$

Open shop -ongelmassa jokainen työ on prosessoitava jokaisella $m$ koneella, mutta prosessointiaika voi olla 0. Töiden reitittämisessä eri koneiden välillä ei ole minkäänlaisia rajoituksia, vaan jokaisella työllä voi olla eri reitti [31].

## 4.4 Laskennallinen kompleksisuus

*Flow Shop* -ongelma on $NP$-kova ja *Hybrid Flow Shop* -ongelma on myös useimmissa tapauksissa $NP$-kova. Vaikka ongelma rajoitettaisiin kahteen vaiheeseen, joista yhdessä vaiheessa on yksi laite ja toisessa vaiheessa kaksi laitetta, on ongelma $NP$-kova. [33, p. 2]

Kahden vaiheen ongelmista $F2||C_{max}$ on ratkaistavissa polynomisessa ajassa *Johnsonin säännöllä* [31, p. 167]. Ongelma $F3||C_{max}$ on taas jo $NP$-kova [31, p. 171].

Job Shop -ongelma $J2||C_{max}$ voidaan redusoida $F2||C_{max}$ ongelmaksi ja on yksi harvoja Job Shop -ongelmia, joille voidaan löytää polynomisessa ajassa toimiva algoritmi. Muut harvat Job Shop -ongelmat, joille polynomisessa ajassa toimiva algoritmi on olemassa, vaativat yleensä, että kaikki prosessointiajat ovat joko 0 tai 1 [31, p. 190].

Open Shop -ongelma $Om||C_{max}$ on $NP$-kova, kun $m \geq 3$ [31, p. 230].

$NP$-kova ongelma on vähintään yhtä hankala ratkaista kuin $NP$-täydellinen ongelma ja se ei ole ratkaistavissa polynomisessa ajassa ellei $P = NP$. [14, p. 109].

## 4.5 Teoreettiset- ja käytännön ongelmat

Käytännön ongelmat ovat dynaamisia ja laajennettuja versioita klassisista optimointiongelmista:

- Työt saapuvat satunnaisin aikavälein,
- laitteet voivat hajota,
- ja töitä voidaan perua ja töiden luvatut toimitus- tai valmistumisajat voivat muuttua [24].
- Lisäksi jos agentti poikkeaa suunnitelmasta suorituksen aikana, voidaan tarvita korjauksia, jotka voivat koskea myös muita agentteja [20].

## 4.6 Algoritmit

### 4.6.1 Hienokuormitussäännöt ja heuristiikat

Suoritettava operaatio laitteelle voidaan valita säännön/heuristiikan perusteella. Lokaalit säännöt vaativat ainoastaan tietoa koneelle odottavista operaatioista ja globaalit säännöt huomioivat myös muiden laitteiden tilan [29]. Taulukossa 5 on otos Panwalkarin ja Inskanderin luetteloimista erilaisista hienokuormitussäännöistä [29]. Kyseisessä artikkelissa erilaisia luetteloituja sääntöjä on kaiken kaikkiaan 113 kappaletta.

| Sääntö | Kuvaus |
|---|---|
| SI / SIO / SPT | Valitaan työ, jolla on pienin välitön suoritusaika. |
| SIS | Valitaan työ, jolla on pienin kokonaissuoritusaika; sisältäen suoritus- ja asetusajat. |
| LI | Valitaan työ, jolla on suurin välitön suoritusaika. |
| LIS | Valitaan työ, jolla suurin kokonaissuoritusaika; |
| DD | Valitaan työ, jolla on aikaisin luvattu toimitus- tai valmistumisaika. |
| FOPNR | Valitaan työ, jolla on vähiten operaatioita jäljellä. |
| Value | Valitaan työ, jolla on korkein arvo. |
| 1/C | Valitaan työ, jolla on korkein myöhästymisen yksikkökustannus. |
| FIFO | Ensimmäinen sisällä on ensimmäinen ulkona. |

Taulukko 5: Esimerkkejä hienokuormitussäännöistä [29]

### 4.6.2 Branch and Bound -algoritmi

Hienokuormitusongelmaa voidaan kuvata graafilla $G = (V, C \cup D)$, jossa $V$ on töiden operaatioita edustavien solmujen joukko ja jokaisella solmulla $i$ on paino, joka vastaa operaation suoritusaikaa $p_i$. Lisäksi graafissa on aloitussolmu 0 ja lopetussolmu $*$, joiden suoritusajat ovat 0. Graafin kaaret ovat $C$ konjuktiivisia kaaria, jotka vastaavat operaatioiden välisiä järjestysriippuvuuksia ja joukko disjunktiivisia kaaria $D$, jotka vastaavat operaatioita samalla koneella ja ovat suuntaamattomia. [8]

Graafista saadaan aikataulu muuntamalla jokainen disjunktiivinen suuntaamaton kaari suunnatuksi kaareksi, jolloin operaatioiden välinen suoritusjärjestys kiinnittyy. Valinta $S$ disjunktiivisten kaarien suunnasta määrittelee kelvollisen aikataulun jos ja vain jos:

- Jokainen disjunktiivinen kaari on kiinnitetty
- ja lopputuloksena syntyvä graafi $G(S) = (V, C \cup S)$ on asyklinen.[8]

**Algoritmi 1:** Branch and Bound -algoritmi [8, s. 114]

```
PRODECUDE Branch and Bound(γ)
    BEGIN
        Calculate a solution S ∈ Y(γ) using heuristics;
        If C_max(S) < UB THEN UB := C_max(S);
        Calculate a critical path P;
        Calculate a blocks of P;
        Calculate the sets E_j^B and E_j^A
        WHILE there exists an operation i ∈ E_j^v with j = 1, ..., k and v = A, B
         DO
            Delete i from E_j^v
            Fix disjunctions for the corresponding successor s
            Calculate a lower bound LB(s) for node s
            IF LB(s) < UB THEN Branch and Bound(s)
        END
    END
```

### 4.6.3 Simuloitu jäähdytys

Simuloitu jäähdytys (engl. Simulated Annealing) on stokastinen heuristinen algoritmi, joka hakee ratkaisua ratkaisuavaruudesta stokastisen mäenkapuamishaun (engl. hill-climbing) avulla. Simuloitu jäähdytys on suosittu metodi ratkaisemaan suuria, monimutkaisia ja käytännön ongelmia, kuten hienokuormitus, aikataulutus ja kauppamatkustajan ongelma (engl. travelling salesman, TSP). Kuten muut hakualgoritmit, simuloitu jäähdytys voi jäädä pyörimään lokaaliin minimiin tai järkevän ratkaisun löytyminen kestää liian kauan.[6].

Simuloitu jäähdytys voidaan nähdä stokastisena päätöksentekoprosessina, jossa *lämpötilaparametria* käytetään epäedullisen päätöksen hyväksyntätodennäköisyyden määrittämiseen. Jos $s_n$, $s'_n$ ja $s_{n+1}$ merkitsevät ratkaisua, johon siirryttiin $n$:llä iteraatiolla, mahdollista ratkaisua johon siirryttäisiin $n$:n iteraation jälkeen ja ratkaisua iteraatiolla $n+1$, niin uusi hyväksytty ratkaisu voidaan määrittää seuraavasti:

$$s_{n+1} \leftarrow \begin{cases} s'_n, & \Delta s < 0 \\ s'_n, & e^{\Delta s/t_n} > \rho \\ s_n, & \text{muutoin.} \end{cases}$$

Tässä $\Delta s = s'_n - s_n$ ja $\rho$ on satunnaisluku, joka on luotu stokastista päätösta varten, ja $t_n$ on lämpötila $n$:llä iteraatiolla. Lämpötila jäähtyy optimoinnin aikana jäähdytysfunktion perusteella $T = f(t_n)$ [6].

Jotta uusi ratkaisu ($s'_n$) hyväksyttäisiin seuraavalle iteraatiolle, sen täytyy joko olla parempi kuin vanha ($s_n$) tai stokastisen säännön tulee täyttyä. Stokastinen sääntö, jonka tarkoitus on todennäköisyyksiin perustuvan päätöksenteon avulla estää optimointiprosessia juuttumasta paikalliseen minimiin, on simuloidun jäähdytyksen perusidea. Todennäköisyys vanhaa ratkaisua huonomman ratkaisun hyväksymiseksi iteraatiolle $n+1$ on $e^{\Delta s/t_n}$. Kun lämpötila laskee $T$:llä, laskee suuren ratkaisua huonontavan askeleen todennäköisyys kohti nollaa Boltzmannin jakauman mukaisesti. Siksi lopullinen ratkaisu on lähellä optimaalista, kun lämpötila lähestyy nollaa [6].

Pseudokoodi simuloidulle jäähdytykselle hienokuormitukseen löytyy listauksesta algoritmi 2. Tässä listauksessa $\beta_k$ on lämpötila iteraatiolla $k$, $G$ on minimoitava tavoitefunktio ja $P(S_c, S_k)$ on todennäköisyys epäedulliseen seuraajaan siirtymisestä $e^{\Delta s/t_n}$.

**Algoritmi 2:** Simuloitu jäähdytys [31, s. 386]

Askel 1:
$k \leftarrow 1$ ja valitse $\beta_1$
Valitaan alustava sekvenssi $S_1$ käyttäen jotain heuristiikkaa.
Asetetaan $S_0 = S_1$
Askel 2:
Valitaan kandidaatti aikataulu $S_c$ $S_k$:n naapurustosta. Jos $G(S_0) < G(S_c) < G(S_k)$, aseta $S_{k+1} = S_c$ ja jatka askeleesta 3.
Jos $G(S_c) < G(S_0) < G(S_k)$, aseta $S_0 = S_{k+1} = S_c$ ja jatka askeleesta 3.
Jos $G(S_c) > G(S_k)$, generoidaan satunnaisluku $U_k$ tasajakaumasta välillä (0,1)
Mikäli $U_k \leq P(S_k, S_c)$ asetetaan $S_{k+1} = S_c$ ja muutoin $S_{k+1} = S_k$ ja jatketaan askeleesta 3.
Askel 3:
Valitaan $\beta_{k+1} \leq \beta k$.
$k \leftarrow k + 1$
Jos $k = N$ niin LOPETA, muutoin jatka askeleesta 2.

### 4.6.4 Geneettiset algoritmit

Geneettinen algoritmi sovellettuna hienokuormitukseen käsittelee yksittäistä aikataulua yksilönä ratkaisupopulaatiossa. Jokaista ratkaisujoukon jäsentä kuvaa sen sopivuus, joka määritellään sopivuusfunktiolla. Algoritmi toimii iteratiivisesti ja jokainen iteraatio on sukupolvi. Ratkaisupopulaation yksi sukupolvi koostuu edellisen sukupolven selviytyjistä ja edellisen sukupolven jälkeläisistä. Ratkaisupopulaation koko on yleensä vakio sukupolvesta toiseen. Jälkeläiset luodaan lisääntymisen ja yksilöiden mutaatioiden kautta edellisestä sukupolvesta. Yksilöihin voidaan myös viitata termillä kromosomi. Jos optimointiongelma koostu aikataulutuksesta useammalle koneelle, voi yksittäinen kromosomi koostua useista alikromosomeista, jotka sisältävät informaatiota töiden järjestyksestä per kone. Jokaisesta sukupolvesta sopivimmat yksilöt lisääntyvät ja vähiten sopivat kuolevat. [31, s. 385].

**Algoritmi 3:** Geneettinen algoritmi [31, s. 386]

Askel 1:

$k \leftarrow 1$

Valitaan $l$ alustavaa sekvenssiä $S_{1,1}, ...S_{1,l}$ jollain heuristiikalla.

Askel 2:

Valitaan kaksi parasta aikataulua joukosta $S_{k,1}, ..., S_{k,l}$
ja merkitään näitä symboleilla $S_k^+$ ja $S_k^{++}$.
Valitaan kaksi huonointa aikataulua joukosta $S_{k,1}, ..., S_{k,l}$
ja merkitään näitä symboleilla $S_k^-$ ja $S_k^{--}$.
Luodaan kaksi jälkeläistä $S^*$ ja $S^{**}$ vanhemmista $S_k^+$ ja $S_k^{++}$.
Korvataan $S_k^-$ ja $S_k^{--}$ jälkeläisillä $S^*$ ja $S^{**}$.
Pidetään muut aikataulut ennallaan ja jatketaan kohdasta 3.

Askel 3:

$k \leftarrow k + 1$

Jos $k = N$ niin LOPETA, muutoin jatka askeleesta 2.

### 4.6.5 Tabu-haku

Tabu-haku (engl. Tabu Search) alkaa alustavalla ratkaisulla, joka talletetaan nykyiseksi tilaksi ja sen hetkiseksi parhaaksi ratkaisuksi. Nykyisen tilan naapurit tuotetaan naapurirakenteen avulla ja nämä ovat uusia ratkaisukandidaatteja. Ratkaisukandidaateille evaluoidaan tavoitefunktio, ja ratkaisukandidaateista valitaan seuraavaksi tilaksi kandidaatti joka ei ole tabu ja täyttää tavoitekriteerit (engl. Aspiration Criterion). Tätä seuraavan tilan valintaa kutsutaan siirroksi ja se tallennetaan tabu-listaan. Mikäli uusi tila on parempi kuin nykyinen paras ratkaisu, se tallennetaan uudeksi parhaaksi ratkaisuksi. Iteraatioita toistetaan kunnes pysähtymiskriteeri on saavutettu [16].

**Algoritmi 4:** Yleinen tabu-haku-algoritmi [16]

Askel 1:

Aloitetaan alustavalla ratkaisulla ja tallennetaan se nykyiseksi sekä parhaaksi ratkaisuksi.

Askel 2:

Luo nykyisen ratkaisun naapuriratkaisut naapurirakenteen avulla.

- Valitse naapuri, joka ei ole tabu ja joka täyttää annetut tavoitekriteerit ja aseta se nykyiseksi ratkaisuksi.
- Päivitä tabu-lista.
- Jos valittu ratkaisu tuottaa paremman arvon tavoitefunktiolle kuin paras ratkaisu, aseta se parhaaksi ratkaisuksi.

Askel 3:

Toista askelta 2, kunnes lopetusehto täyttyy.

Tabu-haun elementit suhteessa hienokuormitusongelmiin (JSP): [16]:

- *Alustava ratkaisu* voidaan luoda erilaisilla mekanismeilla ja sen valinta vaikuttaa hienokuormitusratkaisun laatuun. Mitä parempi aloitusratkaisu on, sitä parempi on myös tabu-haun lopputulos.

- *Naapurirakenne* on mekanismi joka luo uuden joukon naapuriratkaisuja nykyisen ratkaisun yksinkertaisella muokkauksella. Jokainen naapuri on tavoitettavissa nykyisestä ratkaisusta siirrolla. Naapuriratkaisu on tärkeä tabu-haun tehokkuudelle, koska tabu-haku etenee iteratiivisesti yhdestä naapurista toiseen ratkaisuavaruudessa. Siksi naapurirakenteen tulee poistaa tarpeettomat ja ei validit ratkaisut mikäli mahdollista.

- *Siirto*: Paras naapuriratkaisu, joka ei ole tabu tai täyttää tavoitekriteerit valitaan uudeksi nykyiseksi ratkaisuksi. Paras naapuriratkaisu on se, joka minimoi tavoitefunktion (esim. $C_{max}$). Jos kaikki naapuriratkaisut ovat tabuja tai mikään naapuri ei täytä tavoitekriteereitä, valitaan nykyiseksi naapuriksi ensin tabulistalle joutunut naapuri.

- *Tabu-lista ja sen päivitys*: On olemassa kaksi yleistä tapaa muistin käyttöön tabu-haussa: lyhyt- ja pitkäkestoinen muisti. Molempia voidaan pitää tapoina ratkaisun naapurien muokkaukseen. Lyhytkestoinen muisti seuraa ratkaisujen ominaisuuksia, jotka ovat vaihtuneet ja käyttää sitä tabu-listana.

- *Tavoitekriteerit*: kriteerien tavoitteena on naapuriratkaisun tabu-statuksen ohittaminen mikäli tarpeellista. Esim. jos siirto tuottaa ratkaisun, joka on parempi kuin paras tähän mennessä havaittu ratkaisu, siirto suoritetaan vaikka se olisi tabu.

- *Lopetusehto*: Kun ratkaisua huonontavien siirtojen määrä saavuttaa asetetun raja-arvon (esim. 2000) tai uusia naapureita ei voida luoda, tai ratkaisu, jota ei voida toteuttaa havaitaan, Tabu-haku algoritmi päättyy.

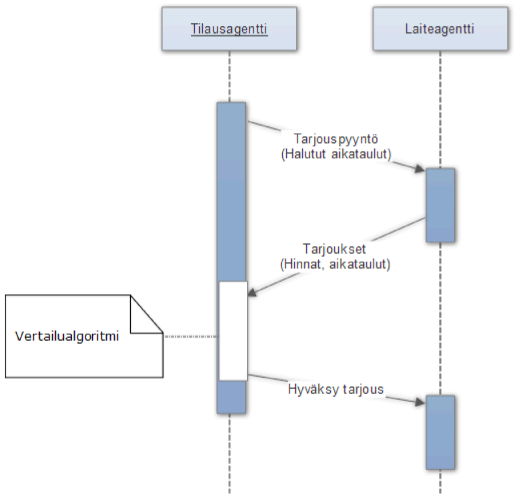
## 4.7 Agenttipohjainen hienokuormitus

Agenttipohjaisessa hienokuormituksessa neuvotteluun perustuvat menetelmät ovat olleet pääsääntöinen ratkaisutapa. Itse neuvotteluprosessi voi olla joko tilauksen tai resurssin käynnistämä ja yleensä työt kuulutetaan yksitellen yleensä edellisen operaation suorittamisen jälkeen [28, p. 705].

Kuvassa 7 on yleiskuva tarjous- ja hinnoitteluprosessista. Tilausagentti (engl. Job Agent) lähettää tarjouspyynnön laiteagenteille (engl. Machine Agent) ja valitsee valinta-algoritmillaan parhaan tarjouksen. Tilausagentti antaa työn parhaan tarjouksen tehneen laiteagentin suoritettavaksi.

Markkinamekanismin huonoina puolina on se, ettei systeemille voida antaa takeita suorituskyvystä ja että on hankalaa taata, ettei äärimmäisiin tilanteisiin jouduttaisi. Tämä johtuu siitä, että markkinamekanismi on perimmiltään epädeterministinen ja hyöty-päättelyyn pohjautuva metodi [28, p. 705].

Markkinamekanismiin perustuvat ratkaisut eivät suoriudu yhtä hyvin kuin globaalit optimointimenetelmät täysin deterministisessä ongelman ratkaisussa. Hajautettu hienokuormitus voi kuitenkin olla tärkeässä roolissa monimutkaisessa laiteympäristössä, jossa esiintyy erilaisia satunnaistekijöitä [31, s. 414].




Kuva 7: Yleiskuva tarjous- ja hinnoitteluprosessista [31, s. 408]

### 4.7.1 Esimerkkejä agenttipohjaisista ratkaisuista

A. Márkus ja kumppanit [26] kuvaavat markkinapohjaisen ratkaisun, jossa erillinen hallinta-agentti ottaa tilaukset vastaan ulkopuolisista järjestelmistä ja neuvottelee töiden suorituksesta laiteagenttien kanssa.

Lin ja Solberg [23] kuvaavat agenttipohjaisen ratkaisun joustavaan tuotannon reitittämisen ja ohjauksen simulointiin. Järjestelmän peruspiirteitä ovat:

- Ohjaus ja hienokuormitus on mallinnettu kauppapaikkana.
- Jokainen systeemin entiteetti on ohjelmistoagentti ja neuvottelee muiden agenttien kanssa.
- Tosiaikainen simulaatio mallintaa agenttipohjaista ja reaaliaikaista hienokuormitusjärjestelmää.
- Jokaista tuotettavaa osaa mallinnetaan omalla agentillaan, joka neuvottelee resurssiagenttien eli esim. laiteagenttien kanssa.
- Agentit reagoivat tapahtumiin (events)

Mařík, Vrban ja Fletcherin MAST tuotantosimulaatio [25] on kehitetty Rockwellin Prahan tutkimuskeskuksessa ja sen ideana on ollut toteuttaa agenttipohjainen ratkaisu erityisesti materiaalien ja tuotteiden siirtämiseen tuotantosoluilta toiselle kuljettimien sekä automaattitrukkien avulla.

- Simulaatio on rakennettu JADE alustalle.
- Agentit toimivat yhteistyössä toistensa kanssa lähettämällä viestejä.
- Kuljettimilla on oma kustannusperustainen malli, jossa jokainen kuljetin tarjoaa kuljetuspalvelua ennalta määritellyllä hinnalla.
- Pääpaino on ongelmien havaitsemisessa ja niistä toipumisessa.

Yksi vaihtoehto on ajaa agentti-ohjelmia ohjelmoitavassa logiikassa [5]. Antzoulatoksen ja kumppanien agenttipohjainen järjestelmä on rakennettu JADE-sovelluskehyksen ympärille ja sisältää seuraavan tyyppisiä agentteja:

1. Tuoteagentti (Product Agent) luodaan, kun operaattori aloittaa tuotteen valmistuksen. Tuoteagentti sisältää kokoamiseen vaaditut tehtävät, kuten osan asentamisen oikealle paikalleen.
2. Systeemiagentti (System Agent) liittää tehtävät järjestelmän tuotannonhallinta-agenteille.
3. Tuotannonhallinta-agentit (Production Management Agent) vastaavat järjestelmän osaamia monimutkaisia tuotanto-operaatioita. Tuotannonhallinta-agentti ryhmittelee ne komponenttiagentit, jotka toimivat yhdessä yhdeksi monimutkaiseksi kokonaisuudeksi.
4. Komponenttiagentit (Component Agent) vastaavat järjestelmään liitettyjä robotteja ja laitteita ja rekisteröivät itsensä tuotannonhallinta-agenttiin.

# 5 Kokeellinen tutkimus: agenttipohjainen hienokuormitus

Tässä luvussa kuvataan kokeellinen tutkimusmenetelmä ja diplomityön tuloksena rakennettu agenttipohjainen tuotantosimulaatio. Luvun lopussa esitellään kokeellisen tutkimuksen lopputuloksena saadut tulokset, jossa verrataan markkinamekanismiin pohjautuvan agenttipohjaisen hienokuormituksen suorituskykyä agenttipohjaiseen keskitettyä optimointia käyttävään menetelmään.

## 5.1 Agenttialustan valinta

Erilaisia agenttialustoja on kymmeniä [19], joista taulukossa 6 on ryhmitelty agenttialustoja käytetyn ohjelmointikielen perusteella. Karavarin ja Bassiliadeksen agenttialustojen vertailun mukaan JADE alusta on suosituin FIPA-yhteensopiva agenttialusta[19, s. 5]. JADE-alustan valintaan vaikuttivat eniten saatavilla oleva dokumentaatio, tuttu ohjelmointikieli (JAVA) sekä alustan helppokäyttöisyys. Lisäksi alustasta oli olemassa oleva oppikirja *Developing Multi-Agent Systems with JADE*[7].

| Kieli | Alusta |
|---|---|
| Java | JADE, SeSAm, Jadex, JAS, AgentBuilder, EMERALD, Repast, MaDKit, CybelePro, Cormas, AGLOBE, Cougaar, Swarm, MASON, INGENIAS Development Kit, AnyLogic, JAMES II |
| C/C++ | AgentBuilder, Swarm (Objective C), Repast (plus C#), MaDKit |
| declarative/rule programming | Repast (Lisp, Prolog), JADE (JESS), EMERALD (JESS) |
| Python | Repast, MaDKit |
| AgentSpeak | Jason, Agent Factory (AFSE) |
| Smalltalk | Cormas |
| JAL | JACK |
| NetLogo | NetLogo |
| GAML | GAMA |
| XML | AgentScape, EMERALD, INGENIAS Development Kit, JACK, Jadex, JIAC |
| Multiple languages | AgentScape, EMERALD, INGENIAS Development Kit, JACK, Jadex, JIAC, Jason, MaDKit, Repast, AgentBuilder, AgentFactory |

Taulukko 6: Agenttialustat ohjelmointikielittäin [19, s. 14]

## 5.2 Toteutus

### 5.2.1 Arkkitehtuuri

Agenttisimulaatio on toteutettu Java-ohjelmointikielellä käyttäen Java SDK 1.8.0 versiota sekä JADE-agenttialustan[18] versiota 4.5. Toteutuksen kehitysympäristönä toimi Eclipse ja projektin build-järjestelmänä Gradle.

JADE-alusta on FIPA-yhteensopiva, jolloin koko järjestelmän arkkitehtuuri vastaa FIPA:n referenssimallia. JADE kattaa toteutuksen FIPA:n referenssimallin mukaisista osista kaikki paitsi yksittäisten agenttien käyttäytymisen toteutuksen.

Kuvassa 8 on toteutetun agenttisimulaation pakettikaavio. Toteutettu osuus on kaksijakoinen, jossa agenttien toteutus on omassa paketissaan ja domain-käsitteet ovat omassaan. Agentit käyttävät domain-olioita toimintalogiikassaan ja lisäksi domain-oliot toteuttavat JADE:n ontologioiden vaatimat rajapinnat, jolloin ne voidaan sarjallistaa ACL-viestien (Agent Communication Language) sisällöksi.

Tärkeimmät käytetyt JADE:n paketit ovat:

- Paketti *jade.core* sisältää JADE:n ytimen, kuten JADE:n agenttien yliluokan *Agent*. Toteutuksen agentit ovat periytetty JADE:n agenttiluokasta.
- Paketti *jade.proto* sisältää JADE:n interaktioprotokollia agenteille. Osa agenttien käytöksestä (*behaviours*) on periytetty tässä paketissa määritellyistä protokollista.
- Paketista *jade.domain* on käytetty luokkia mm. hakupalvelun (*Directory Facilitators / yellow pages*) käyttöön.
- Paketti *jade.lang.acl* sisältää luokkia agenttien väliseen kommunikointiin (*Agent Communication Language*).
- Paketti *jade.wrapper* sisältää luokkia itse JADE-alustan kontrollointiin ja näitä käytetään simulaation pystytyksessä esim. agenttien luomiseen.
- Paketit *jade.content* ja *jade.content.onto* sisältävät luokkia ontologioiden luomiseen. Simulaatiossa on luotu yksinkertainen ontologia, jota käytetään ACL-viesteissä agenttien väliseen kommunikointiin.

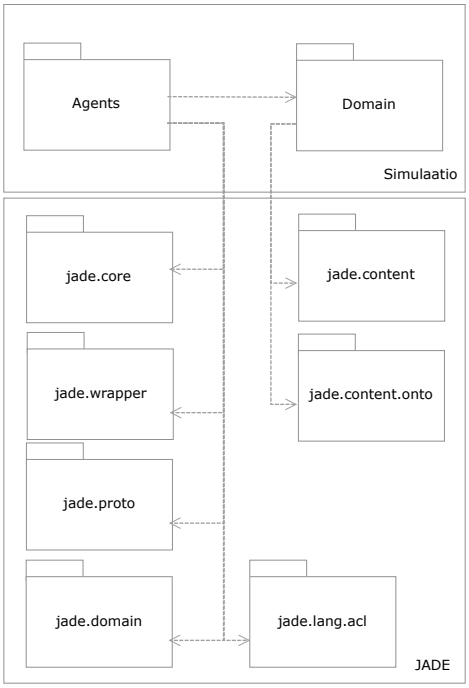
| Työkalu | Nimi | Versio |
|---|---|---|
| Ohjelmointikieli | Java | OpenJDK 1.8.0 |
| Agenttialusta | JADE | 4.5 |
| Kehitysympäristö (IDE) | Eclipse | Oxygen.1 Release (4.7.1) |
| Koontityökalu | Gradle | 2.13 |
| Käyttöjärjestelmä | Fedora (Linux) | 25 |

Taulukko 7: Toteutuksessa käytetyt työkalut

### 5.2.2 Agenttipohjainen suunnitteluprosessi

MAST case-studyssä[25] luetellaan agenttipohjaisen suunnitteluprosessin vaiheita:

1. *Agenttien tunnistaminen*: agenttipohjaisen järjestelmän suunnittelu alkaa ohjattavan tuotantojärjestelmän analyysistä(i) sekä (ii) tuotannon asettamista vaatimuksista, rajoitteista plus saatavilla olevista laitteistoista ja ohjelmistoista. Tämän analyysin perusteella syntyy lista agenttien tyypeistä, joita järjestelmässä tarvitaan. Yleinen suunnitteluperiaate - joka seuraa oliopohjaisia menetelmiä




Kuva 8: Pakettikaavio toteutuksesta

- on, että jokainen laite, kuljetin, tuotantosolu jne. esitetään erillisenä agenttina. Myös tuote itsessään tai puolivalmiste voidaan esittää omana agenttinaan, joka neuvottelee sen oman tuotannon/kokoamisen muiden tuotantoympäristön agenttien kanssa.

2. *Implementointi*: Tunnistetut agenttityypit/-luokat toteutetaan käyttäen hyväksi agenttikirjastoa, joka joko kehitetään alusta alkaen tai käytetään hyväksi olemassa olevaa kirjastoa. Agentit luodaan instansseina agenttikirjastossa olevien agenttiluokkien perusteella. Lisäksi kommunikaatio agentti-instanssien välillä määritellään ohjelmistokehyksen alustuksessa geneeristen agenttiluokkien välillä. Näin tuotetaan ensimmäinen prototyyppi agenttipohjaisesta ohjaus-/tuotantojärjestelmästä.

3. *Simulaatio*: Agenttipohjaisten järjestelmien käyttäytyminen on ennemmin vuorovaikutuksesta ilmaantuvaa (engl. emergent) kuin determinististä. Päätöksentekoon tarvittava tieto on lokaalisti tallennettu agentteihin agenttienvälisen kommunikointimallien lisäksi. Agenttien välisestä vuorovaikutuksesta syntyy järjestelmän yleinen käyttäytyminen, jota ei voida tarkasti ennakoida etukäteen. Koska fyysisten laitteiden kanssa tehtävä testaus tuotantoympäristössä on kallista eikä lähtökohtaisesti realistinen vaihtoehto, on simulaatio ainoa tapa.

4. *Kohteena olevan tuotanto-/ohjausjärjestelmän implementaatio*: tässä vaiheessa järjestelmä toteutetaan lopullisessa järjestelmässä ajettavaksi koodiksi. Toteutus riippuu usein logiikoista, strukturoidusta tekstistä jne. alhaisimman

tason ohjausjärjestelmistä. Korkeamman tason ohjaus - joka on toteutettu agenteilla - voidaan lähes kokonaan käyttää uudelleen, esim. agentit simulaatiojärjestelmästä ovat käytettävissä myös varsinaisen tuotantojärjestelmän ohjaukseen.

Kohdat 1, 2 ja 3 ovat oleellisia koeasetelman kannalta. Vaatimukset ja rajoitteet tulevat valitun hienokuormitusongelman määrittelystä, valitusta agenttialustasta ja MES-näkökulmasta. Implementointiin taas vaikuttaa lähinnä valittu agenttialusta (JADE). Kohdalla 4 ei ole suoranaista vaikutusta toteutettavaan simulaatioon, mutta lopputulosten pohdinnassa voidaan miettiä myös minkälaisia muutoksia tarvittaisiin oikean tuotantojärjestelmän rakentamisessa.

### 5.2.3 Agenttien roolit

Agenttipohjaisen suunnitteluprosessin (kappale 5.2.2) perusteella työssä tunnistettiin seuraavat simulaatiossa käytettävät agenttityypit:

- *Simulaatioagentti* hoitaa simulaation käynnistyksen ja tulosten kirjaamisen. Agentti alustaa tarvittavat tilaus-, laite- ja optimointiagentit. Simulaatioagentti lopettaa sovelluksen, kun kaikki tilausagentit ovat valmiita. Tilausagentti kuuntelee laiteagenttien tilaa ja kirjaa tilamuutokset tiedostoon analysointia varten.

- *Tilausagentti* mallintaa järjestelmään saatua tuotantotilausta ja välittää tilauksen operaatiot laite- ja/tai optimointiagenteille suoritettavaksi. Kun tilausagentti on saanut tilauksen kaikki operaatiot suoritetuksi, se lopettaa toimintansa.

- *Laiteagentti* saa suoritettavat operaatiot joko tilausagentilta tai optimointiagentilta. Kun laiteagentti on saanut operaation suoritettua, se ilmoittaa suorituksen onnistumisesta tilaus-/optimointiagentille.

- *Optimointiagentti* on olemassa keskitettyä hienokuormitusta varten ja lähettää tilausagenteilta saamansa operaatiot suoritettavaksi laiteagenteille valitun optimointialgoritmin mukaisessa järjestyksessä. Toteutettu agentti optimoi tuotantoa geneettisellä algoritmilla.

### 5.2.4 Agenttien toteutus

Valitussa JADE-ohjelmistokehyksessä agentit toteutetaan periyttämälle ne JADE:n *jade.core.Agent*-luokasta, joka on yleinen yliluokka kaikille JADE:n ohjelmistoagenteille. Agenttiluokka tarjoaa toteutuksen viestien lähettämiseen ja agentin tilan hallintaan, kuten agentin suorituksen aloittamiseen, keskeyttämiseen ja lopettamiseen.

Agenttien käyttäytyminen määritellään agentteihin liittyvillä abstraktista *jade.core.behaviours.Behaviour*-luokasta periytetyillä luokilla, jotka mallintavat agenttien käytöstä. JADE-agentilla on yksi säie, jossa agentin toiminta suoritetaan, mutta

liittämällä agenttiin useampi käytösluokka, voidaan mallintaa useampaa yhtäaikaista toiminnallisuutta.

Tehdyssä toteutuksessa seuraavat luokat on periytetty JADE:n *jade.core.Agent*-luokista ja toteuttavat seuraavanlaiset käytökset:

- *Simulaatioagentti SimulationAgent* sisältää käytöksen *LoggingBehaviour*, joka on JADE:n *TickerBehaviour*-luokasta periytetty toiminnallisuus. *TickerBehaviour* käynnistää toteutuksen säännöllisin väliajoin ja *LoggingBehaviour* tarkistaa säännöllisin väliajoin simulaatioagentin vastaanottamat viestit. Viestit kirjataan tulostiedostoon ja mikäli kaikki tilausagentit ovat poistaneet rekisteröintinsä hakemistopalvelusta, niin simulointi lopetetaan.

- *Tilausagentti OrderAgent* sisältää käytökset *ManufacturingOrderBehaviour* ja *NegotiateManufacturerBehaviour*. *ManufacturingOrderBehaviour* on *TickerBehaviour*, joka säännöllisin väliajoin tarkistaa onko neuvotteluprosessi käynnissä tilauksen operaatiolle tai onko koko tilaus valmis. Mikäli neuvotteluprosessi ei ole käynnissä, käynnistää *ManufacturingOrderBehaviour* uuden *NegotiateManufacturerBehaviour* käytöksen seuraavalle operaatiolle.

  *NegotiateManufacturerBehaviour* on *ContractNetInitiator*-luokasta periytetty käytös, joka toteuttaa FIPA ContractNet -protokollasta neuvottelun aloittajan osan. Käytös lähettää tarjouspyynnön jokaiselle hakupalveluun rekisteröidylle tuottajalle ja hyväksyy edullisimman tarjouksen.

- *Laiteagentti MachineAgent* sisältää käytökset *ManufacturingBehaviour* ja *NegotiateBehaviour*. *ManufacturingBehaviour* on *TickerBehaviour*, joka simuloi tuotantoa. Käytös valitsee agentin hyväksyttyjen tarjousten jonosta yhden suoritettavaksi, hylkää muut ja ilmoittaa suorituksen päätyttyä onnistumisesta *tilausagentille* sekä *simulaatioagentille*.

  *NegotiateBehaviour* on JADE:n *SSContractNetResponder*-luokasta periytetty toiminnallisuus, joka vastaa yhden tarjouspyynnön tarjousprosessista. Käytös lähettää vastineena tarjouspyyntöön kustannuksen, joka valituissa simulaatioissa on tuotannon lopetusaika. Hyväksytyt tilaukset tallennetaan agentin hyväksyttyjen tilausten jonoon.

  *OptimizerListenerBehaviour* on optimoijan kanssa käytettävä käytös, joka lisää kaikki optimoijan lähettämät tehtävät suoraan agentin työjonoon.

- *Optimointiagentti OptimizingAgent* sisältää käytöksen *OptimizedManufacturingOrderBehaviour*, joka lähettää tilausten osat suoritettavaksi laiteagenteille. Mikäli osatilaus epäonnistuu jostain syystä, lähettää optimointiagentti sen uudelleen suoritukseen. Mikäli suunniteltua agenttia ei ole saatavilla, valitsee optimointiagentti korvaavan agentin aikaisimman valmistumisajan perusteella.

### 5.2.5 Käytetty optimointialgoritmi

Toteutuksessa käytetty optimointialgoritmi vastaa artikkelissa *Hybrid Flow Shop Scheduling Using Genetic Algorithms*[39] määriteltyä algoritmia. Algoritmi etsii

sopivaa tilausten suoritusjärjestyksen permutaatiota, jossa aloitusvaiheen jälkeen seuraavissa vaiheissa tilaukset käsitellään FIFO-järjestyksessä. Sopivuusfunktiota varten algoritmi simuloi jokaiselle populaation jäsenelle tuotannon ja laskee käytetyn tuotantoajan tilausjoukolle ($C_{max}$).

Geenien esitysmuotona on taulukko, jossa jokainen arvo vastaa tilausnumeroa ja järjestys taulukossa on tilausten suoritusjärjestys esim. [3 2 1] olisi tilausten suoritus käänteisessä järjestyksessä suhteessa tilausnumeroon.

Risteytysoperaatio valitsee kahden vanhemman geeneistä leikkauskohdan ja tuottaa kaksi jälkeläistä joissa leikkauskohdan kohdalla vaihtuu kummasta vanhemmasta geenit ovat peräisin. Jotta jälkeläiset olisivat kelvollisia, korjataan suoritusjärjestys arpomalla uusi käyttämätön arvo useammin kuin kerran esiintyvien arvojen tilalle.

Mutaatio-operaattorina toimii kahden geenin vaihtaminen satunnaisesti päittäin. Operaation etuna on, että mutatoitunut geeni on aina validi permutaatio.

### 5.2.6 Hakemistopalvelun käyttö

Agentit rekisteröivät itsensä JADE-ympäristön hakupalveluun, jotta muut agentit löytävät niiden tarjoamat palvelut. Laiteagentit rekisteröivät palvelun, josta selviää laitteen tyyppi eli Hybrid Flow Shop -ongelman tapauksessa laitteen vaiheen numero. Optimointi- ja tilausagentit rekisteröivät itsensä palveluun ja poistavat itsensä palvelusta kun ovat saaneet suorituksensa valmiiksi.

Esimerkki laiteagentin rekisteröitymisestä palveluun, jossa ensin luodaan agentin ja palvelun kuvaus, jotka sitten rekisteröidään:

```
m_template = new DFAgentDescription();
m_template.setName(this.getAID());
ServiceDescription sd = new ServiceDescription();
sd.setName("manufacturing");
sd.setType("equipment-type-"+ m_equipment.getEquipmentType());
sd.setOwnership(m_equipment.getId());
m_template.addServices(sd);
try {
   DFService.register(this, template);
```

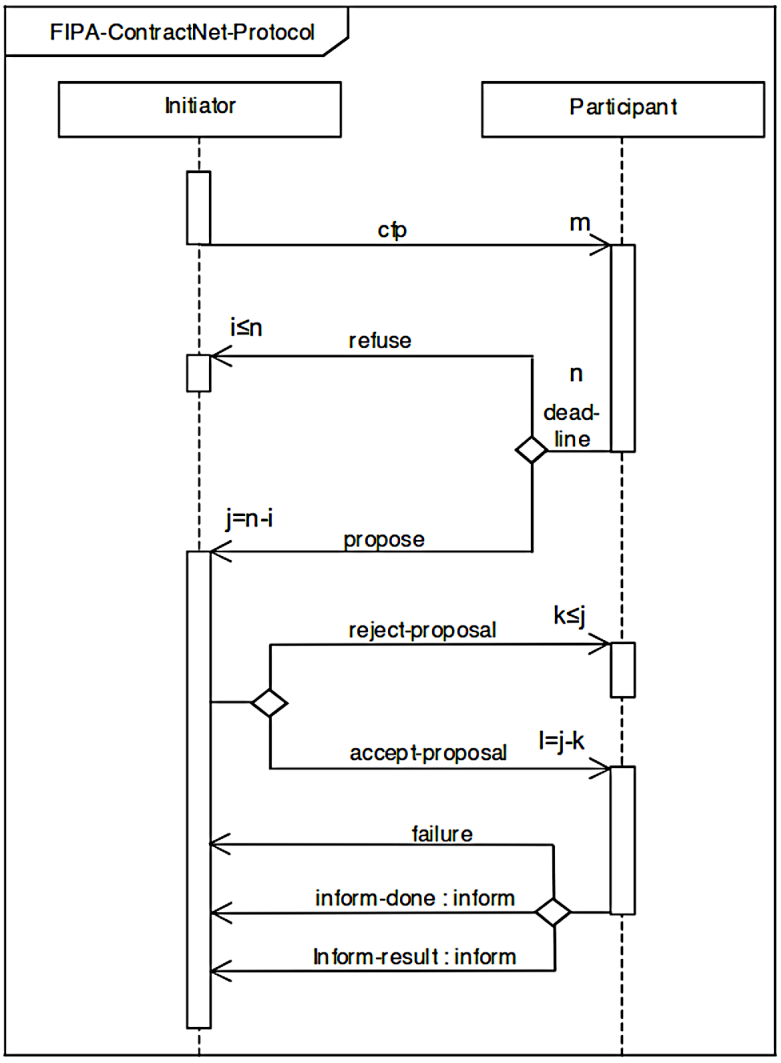
Vastaavasti agentin poisto palvelusta takeDown-metodissa, jota JADE-alusta kutsuu agenttia sammutettaessa:

```
protected void takeDown()
{
    super.takeDown();
    System.out.println("Optimizer-agent " + getAID().getName() + "finished.");
    try {
        DFService.deregister(this);
    } catch (FIPAException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

### 5.2.7 Agenttien kommunikointi

JADE-ympäristössä agentin kommunikoivat toistensa kanssa viestinvälityksen avulla. JADE tarjoaa kaiken infrastruktuurin agenttien väliseen kommunikointiin ja lisäksi tuen protokollien toteuttamiseen. JADE:n toiminnallisuuksista agentit käyttävät keskinäiseen kommunikaatioonsa viestinvälitysinfrastruktuuria, hakemistopalvelua ja FIPA Contract Net -protokollaa.

FIPA Contract Net -protokolla on kuvattu kuvassa 9. Protokollassa aloittaja lähettää osallistujille tarjouspyynnön (cfp), johon osallistujat vastaavat deadlinen puitteissa joko hylkäyksellä (refuse) tai tarjouksella (propose). Aloittaja valitsee parhaan tarjouksen ja ilmoittaa valinnasta osallistujille tarjouksen hylkäyksellä (reject-proposal) tai hyväksynnällä (accept-proposal). Hyväksytyn tarjouksen esittänyt osallistuja ilmoittaa työn suorituksesta aloittajalle epäonnistumisena (failure) tai onnistumisena (inform-done).




Kuva 9: FIPA Contract Net -protokolla [13, s. 5].

Toteutetussa ratkaisussa agenttien välistä viestintää varten muodostettiin oma *ontologia*, joka kuvaa hienokuormituksessa tarvittavat käsitteet. Ontologia *Production-Ontology* on rakennettu JADE:n *BeanOntology*-luokasta periyttämällä ja sisältää seuraavat käsitteet:

- *DomainObject* on yläkäsite kaikille domainin käsitteille ja sisältää yksilöivän tunnuksen.
- *Equipment* on käsite suorittaville laitteille sisältäen kilpinopeuden ja laitetyypin.
- *Product* on käsite, joka kuvaa valmistettavaa tuotetta ja sisältää kuvauksen tuottaan valmistuksesta eli tarvittavista laitteista.
- *Order* on käsite, joka kuvaa tilausta ja sisältää tilatun kappalemäärän sekä tilatun tuotteen tiedot.
- *MachineStatus* on käsite, joka kuvaa laitteen tilaa joka tallennetaan eli simulaation tapauksessa nelikko (laite, tilaus, aloitusaika, lopetusaika).
- *RequestForProposal* on predikaatti, joka kuvaa tarjouspyyntöä ja sisältää tiedot tilauksesta sekä pyytävästä agentista.
- *Proposal* on predikaatti, joka kuvaa tarjousta ja sisältää tiedot kustannuksesta, valmistavasta agentista sekä valmistavan agentin aikaleiman tarjoukselle.
- *AcceptProposal* on predikaatti tarjouksen hyväksynnälle, joka sisältää vastineena tarjoukseen tarjouksen aikaleiman.
- *InformStatus* on predikaatti laitteen tilatiedon lähettämiselle ja sisältää lähettävän agentin sekä *MachineStatus* tilatiedon.
- *TaskStatus* on predikaatti tilauksen operaation tilatiedon lähettämiseen; onnistunut suoritus tai epäonnistunut suoritus.

Agentin toimintaa kuvaavat käsitteet sisältyvät FIPA Contract Net-protokollaan, joten periaatteessa viestin sisällöksi riittäisi pelkkä *käsite (engl. concept)*. Ontologioita käytettäessä tämä ei ole kuitenkaan mahdollista, vaan viestin sisältö voi olla joko *predikaatti (engl. Predicate)* tai *agentin toimenpide (engl. Agent Action)*. Simulaatiossa valittiin näistä vaihtoehdoista aina predikaatti, koska viestin sisältö on aina joku fakta vaikkakaan ei varsinaisesti propositio, ja agentin toimenpide on määritelty jo protokollassa.

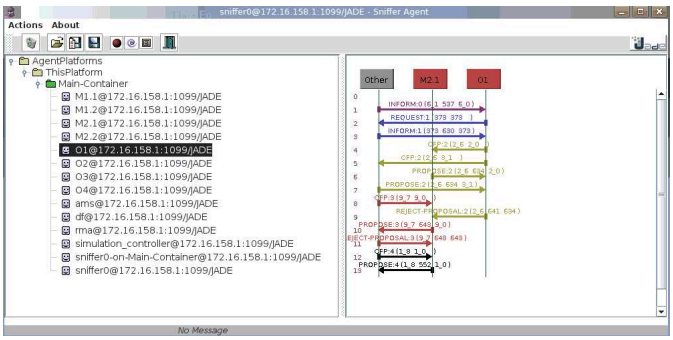
Viestien koodaukseen valittiin JADE:n *SLCodec* joka tulee *FIPA-SLn* kieliä. Koodekin käytön etuna on, että viestit ovat tekstimuotoisena helposti luettavissa. Esim. Contract Net -protokollan tarjouspyyntö käyttäen *ProductionOntoloty*-ontologiaa on seuraavankaltainen:

```
(CFP
:sender  ( agent-identifier :name O24@172.16.158.1:1099/JADE  :addresses (sequence http://mustamursu:7778/acc ))
:receiver  (set ( agent-identifier :name M4.2@172.16.158.1:1099/JADE  :addresses (sequence http://mustamursu:7778/acc )) )
:content  "((RequestForProposal (agent-identifier :name O24@172.16.158.1:1099/JADE :addresses (sequence http://mustamursu:7778/acc))
           (Order :id O24 :endProduct (Product :id P1 :equipmentType (sequence 0 1 2 3 4) :type (ProductType :name FINAL)) :quantity 100)))"
:reply-with  R1510204907901_0  :language  fipa-sl  :ontology  ProductionOntology  :reply-by  20171109T052217896Z
:protocol  fipa-contract-net
:conversation-id  C2076032232_O24_1510204907901_167 )
```

Kuvassa 10 näkyy JADE:n *sniffer*-toiminnolla kaapattua kommunikaatiota agenttien välillä skenaariosta, jossa on kaksi vaihetta, neljä laiteagenttia ja neljä tilausagenttia. *M2.1* on yksi laiteagenteista ja *O1* tilausagentti. Ensimmäisenä kommunikaatiossa näkyy *INFORM:0*-viesti, joka ilmoittaa tilausagentille ensimmäisen vaiheen operaation valmistumisesta. Seuraavaksi tilausagentti pyytää hakemistopalvelulta listaa toisen vaiheen suorittavista agenteista *REQUEST:1* ja saa ne vastauksessa *INFORM:1*.

Tilausagentti *O1* lähettää tarjouspyynnön *CFP:2* laiteagenteille jotka kuuluvat seuraavaan vaiheeseen. Agentti *M2.1* vastaa pyyntöön tarjouksella *PROPOSE*, mutta agentti *O1* hylkää tarjouksen vastauksella *REJECT-PROPOSAL*.

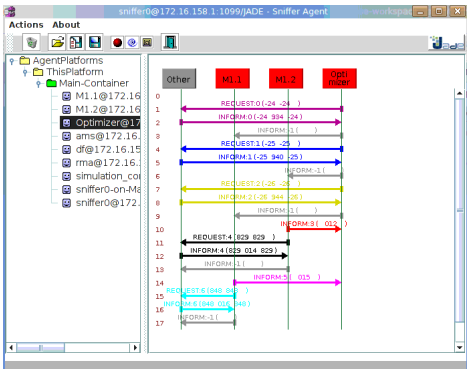
Kuvasta nähdään myös, että ympäristössä on samanaikaisesti useita laiteagentteja $M$ ja tilausagentteja $O$. Laiteagentti $M2.1$ saa näin tarjouspyyntöjä useammalta tilausagentilta tilausten vaiheen 2 suorittamiseksi.




Kuva 10: Agenttien viestinvälitystä JADE-ympäristössä.

Optimointiagenttia käytettäessä kommunikaatio ei tapahdu Contract Net -protokollaa käyttäen, vaan optimointiagentti lähettää tilaukset suoraan suoritettavaksi laiteagenteille. Kuvassa 11 näkyy optimointiagentin kommunikointi laiteagenttien kanssa. Mikäli laiteagentti epäonnistuu operaation suorituksessa tai suunniteltua laiteagenttia ei löydy, lähettää optimointiagentti reaktiivisesti työn suoritettavaksi vaihtoehtoiselle laiteagentille. Vaihtoehtoinen laiteagentti valitaan agentin työjonon ja kilpinopeuden perusteella, joista lasketaan agentin aikaisin mahdollinen valmistumisajankohta.

Kaikki laiteagenttien ja optimointiagentin väliseen kommunikaatioon liittyvät viestit ovat *INFORM*-tyyppisiä. Kuvassa kohteeseen *Other* lähetetyt viestit ovat viestejä hakupalveluun sekä tulokset simulaatioagentille tulosten kirjausta varten.




Kuva 11: Agenttien viestinvälitystä JADE-ympäristössä optimointiagentin kanssa.

### 5.2.8 Lukkiuma

Oletusarvoisesti JADE:n ContractNet-protokollan toteuttavat luokat toimivat niin, että jokainen ContractNetResponder käsittelee kerrallaan ainoastaan yhden tarjouspyynnön ja seuraavan tarjouspyynnön käsittely alkaa vasta ensimmäisen valmistuttua. Jos järjestelmässä on useampi tarjousprosessin aloittaja, ContractNetInitiator, niin eri aloittajat voivat jäädä odottamaan ristikkäisiltä agenteilta tarjousta. Tällöin kumpikaan protokollan mukainen keskustelu ei etene, vaan seurauksena on *lukkiuma* (engl. deadlock).

Ratkaisu *deadlock*-ongelmaan oli koodin muokkaaminen niin, että jokaisesta tarjouspyynnöstä luodaan oma keskustelu *SSContractNetResponder*-luokasta periyttämällä. *SSContractNetResponder* on luokka joka käsittelee vain yhden tarjouspyynnön ja lopettaa sen jälkeen suorituksensa. Jokainen tarjouspyyntö siis käsitellään erillisessä instanssissaan, jolloin ristikkäistä odotusta ei pääse tapahtumaan ContractNetInitiator (tilausagentti) ja ContractNetResponder (laiteagentti) välillä.

### 5.2.9 Tuotannon simulointi

Tuotannon simulointi toteutetaan laiteagentin avulla. Laiteagentilla on ominaisuutenaan kilpinopeus ja laitetyyppi, joista kilpinopeus määrää yhdessä suoritettavan operaation koon (tuotettavat kappaleet) perusteella suorituksen vaatiman ajan. Laiteagentti hoitaa tuotannon simulaation reaaliajassa ja ilmoittaa tuotannon valmistumisesta tilausagentille.

Käytännössä tuotanto jota simuloidaan on *Hybrid Flow Shop*, $\alpha_1 = FHm$ mallista tuotantoa, jossa jokaisessa vaiheessa on useampi rinnakkainen samankaltainen laite $\alpha_3 = Q$. Saman vaiheen laitteet eroavat toisistaan vain mahdollisen kilpinopeuden perusteella, jolloin voi olla edullisinta suosia nopeampia koneita tietyssä vaiheessa tuotantoa. Lisäksi oletetaan, että reititys kaikilta vaiheen $k$ koneilta on mahdollista kaikille vaiheen $k + 1$ koneille.

Mikäli kyseessä olisi oikea tuotantolaitos, asettaisivat mm. seuraavat tekijät todennäköisesti lisärajoitteita tuotannolle, mutta näitä tekijöitä ei simulaatiossa otettu huomioon:

- raaka-aineiden saatavuus,

- puolivalmisteiden saatavuus välivarastoista,
- välivarastojen koko,
- siirtoajat eri prosessivaiheiden välillä,
- asetusajat eri laitteilla,
- tuotevaihtoajat, mahdollisesti riippuen tuotettavista materiaaleista.

## 5.3 Koejärjestelyt

Koeajot suoritettiin kehitysympäristössä, jonka kokoonpano näkyy taulukossa 8. Kokeet ajettiin reaaliaikasimulaationa, jolloin pienet heitot algoritmien suoritusajoissa eivät vaikuta tuloksiin, koska simulaatioaika dominoi. Kuitenkin mikäli algoritmit käyttävät merkittävästi aikaa suoritukseensa, syö algoritmien suoritusaika simuloitua tuotantoaikaa ja vaikuttaa koko järjestelmän tehokkuuteen.

Kokeissa ajettiin JADE:lla toteutettua agenttisimulaatiota seuraavilla skenaarioilla:

1. Kahden koneen ja yhden vaiheen skenaario: $FH1Q2||C_{max}$
2. Hybrid Flow Shop -simulaatio $FHmQm||C_{max}$

| Komponentti | Kuvaus |
|---|---|
| CPU | Intel Xeon E3 1270 3,4 GHz |
| Muisti | 16 Gt DDR3-1333ECC |
| Java | OpenJDK 1.8.0 |
| JADE | 4.5 |
| Fedora (Linux) | 25 |

Taulukko 8: Koeajossa käytetty ympäristö.

Tarkempi kuvaus simulaatioiden ajamisesta löytyy liitteestä B.

### 5.3.1 Tulosten tilastollinen käsittely

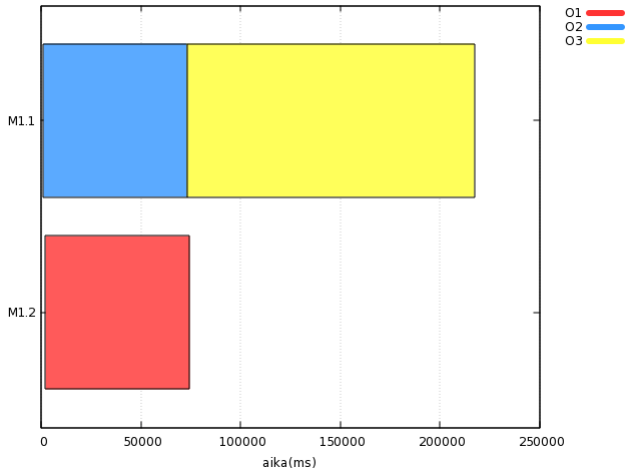
Jokaisessa mittauspisteessä kokeita ajettiin 20 kertaa ja tuloksista laskettiin minimi, maksimi, keskiarvo ja mediaani. Koeympäristön kuormituksesta johtuvat heitot suoritusajoissa eivät ole mielenkiintoisia, koska simulaatioaika dominoi, mutta löydetty aikataulu voi vaikuttaa simulaatioaikaan merkittävästi.

Gantt-kaaviot suoritustuloksista muodostettiin python-skriptillä *gantt.py*[15], joka tuottaa lopputuloksenaan komentosarjat *gnuplot*-ohjelmalle.

## 5.4 Koetulokset

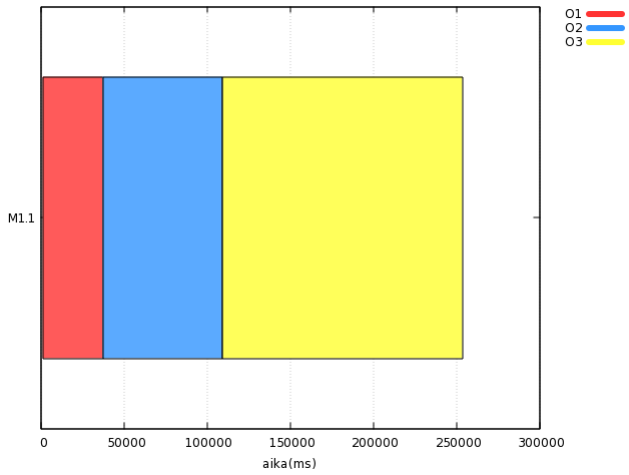
### 5.4.1 Triviaali tapaus

Simuloidaan tuotantoa kahden koneen linjalla, jossa ensimmäinen koneista on kilpinopeudeltaan kaksinkertainen toiseen verrattuna. Mikäli Conract Net -protokollaa käyttävä simulaatio toimii järkevästi, tulisi tilausten painottua ensimmäiselle koneelle. Gantt-kaaviosta kuvassa 12 nähdään, että näin todella käy.




Kuva 12: Tuotantosimulaatio kahdella koneella ja kolmella tilauksella.

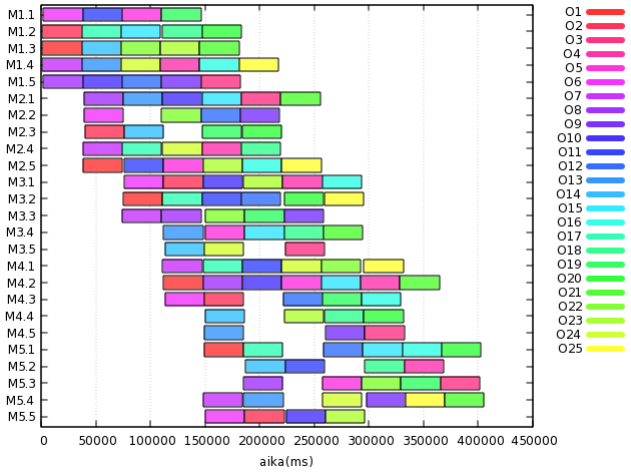
Lisäksi on havaittavissa mielenkiintoinen anomalia, jos pienin tilaus valitaan suoritettavaksi ensin. Silloin kaikki tilaukset suoritetaan laitteella 1 vaikka kokonaisuuden kannalta olisi järkevää käyttää myös laitetta 2. Gantt-kaavio tällaisesta anomaliasta näkyy kuvassa 13.




Kuva 13: Epäoptimaalinen kuormitus yksinkertaisessa tapauksessa.

### 5.4.2 Identtiset tilaukset

Identtisten tilausten tapauksessa jokainen laite ja jokainen tilaus on identtinen. Järkevä suunnitelma on yksinkertaisesti allokoida sama määrä suoritettavia tilauksia jokaiselle laitteelle. Contract Net-simulaation lopputuloksessa havaitaan erikoinen ilmiö kuvassa 14. Aikataulussa on selvästi turhia aukkoja, jotka ovat lisäksi yhden suoritettavan tilausyksikön kokoisia.




Kuva 14: Tuotantosimulaatio 25 tilauksella ja 25 koneella viidessä vaiheessa.

### 5.4.3 Suorituskykyvertailu Hybrid Flow Shop -ongelmalla

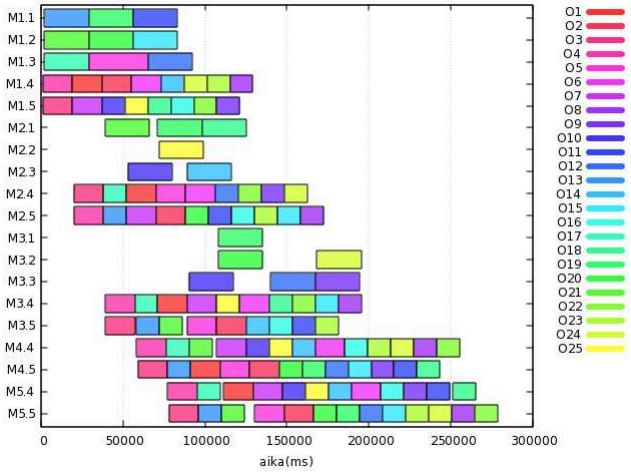
Contract Net -simulaation ja geneettisen algoritmin suorituskykyvertailu toteutettiin ajamalla simulaatio muutamalla erikokoisella ongelmalla. Simuloiduissa tapauksissa tilauksen suoritus valitulla laitteella onnistui aina. Jokaisen vaiheen laitteet oli jaettu kahteen n. samansuuruiseen ryhmään, jossa toisen ryhmän suorituskyky oli kaksinkertainen ensimmäiseen verrattuna. Tilaukset oli jaettu myöskin kahteen ryhmään, jossa ensimmäisen ryhmän tilaukset olivat kooltaan 133% toisen ryhmän tilauksista ja ensimmäiseen ryhmään kuului n. 25% kaikista tilauksista.

Geneettisen algoritmin populaation kokona oli 5000 yksilöä, joista 1000 säilytettiin seuraavalle kierrokselle ja 4000 muodostettiin 1000 parhaan yksilön risteytyksinä. Lisäksi populaatioon joka kierroksella aiheutettiin satunnaisesti 100 mutaatiota. Kierrosten määrä rajoitettiin viiteenkymmeneen (50).

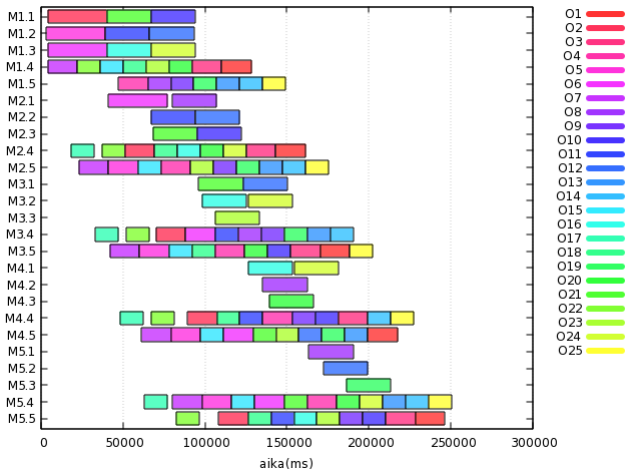
Mittaustulokset näkyvät taulukossa 9: sarakkeessa ongelma on tuotantolinjan konfiguraatio ja tilausten määrää näkyy sarakkeessa $n$. Esimerkit, minkälaisia aikatauluja eri algoritmit tuottavat, löytyvät kuvista 15 (Contract Net) ja 16 (geneettinen algoritmi).

| Ongelma | $n$ | Tulos | Contract Net ($ms$) | GA ($ms$) | $ContractNet/GA$ |
|---|---|---|---|---|---|
| $FH2Q2\|\|C_{max}$ | 4 | MEDIAANI | 102186 | 103201 | 0,99 |
| $FH2Q2\|\|C_{max}$ | 4 | KA | 108590 | 104450 | 1,04 |
| $FH2Q2\|\|C_{max}$ | 4 | MIN | 101170 | 102194 | 0,99 |
| $FH2Q2\|\|C_{max}$ | 4 | MAX | 129219 | 112220 | 1,15 |
| $FH5Q5\|\|C_{max}$ | 25 | MEDIAANI | 289005 | 239474 | 1,21 |
| $FH5Q5\|\|C_{max}$ | 25 | KA | 287961 | 240022 | 1,20 |
| $FH5Q5\|\|C_{max}$ | 25 | MIN | 275483 | 233464 | 1,18 |
| $FH5Q5\|\|C_{max}$ | 25 | MAX | 306522 | 250494 | 1,22 |
| $FH7Q7\|\|C_{max}$ | 49 | MEDIAANI | 420247 | 351720 | 1,19 |
| $FH7Q7\|\|C_{max}$ | 49 | KA | 418716 | 351060 | 1,19 |
| $FH7Q7\|\|C_{max}$ | 49 | MIN | 403749 | 338701 | 1,19 |
| $FH7Q7\|\|C_{max}$ | 49 | MAX | 434800 | 360716 | 1,21 |
| $FH10Q10\|\|C_{max}$ | 100 | MEDIAANI | 644719 | 537145 | 1,20 |
| $FH10Q10\|\|C_{max}$ | 100 | KA | 643147 | 539944 | 1,19 |
| $FH10Q10\|\|C_{max}$ | 100 | MIN | 626187 | 524123 | 1,19 |
| $FH10Q10\|\|C_{max}$ | 100 | MAX | 663221 | 572204 | 1,16 |

Taulukko 9: Suorituskykyvertailu Hybrid Flow Shop -ongelmalla



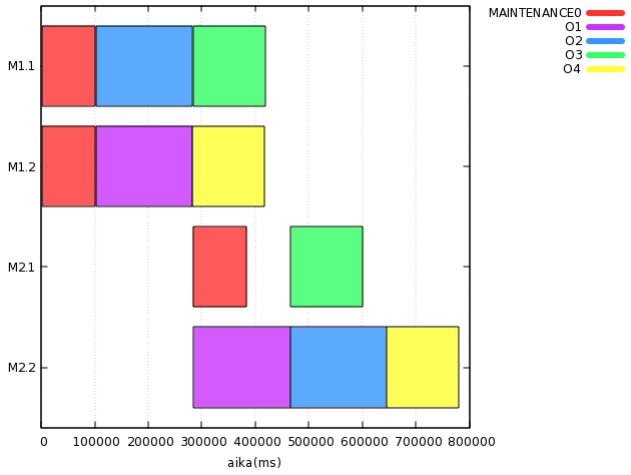

Kuva 15: Esimerkki aikataulusta: Contract Net $FH5Q5||C_{max}$.




Kuva 16: Esimerkki aikataulusta: geneettinen algoritmi $FH5Q5||C_{max}$.

### 5.4.4 Laiterikkojen simulointi

Laiterikkoja simuloitiin niin, että ennalta määrätylle määrälle laitteita arvotaan operaatio, jolla ne rikkoontuvat. Rikkoontuva laite poistaa itsensä huollon ajaksi hakemistopalvelusta, jolloin sille ei voida allokoida uusia operaatiota. Laitteelle jo allokoiduista operaatioista lähetetään tilaus- tai optimointiagentille virheviesti. Kuvassa 17 on esimerkki simulaatiosta, jossa kolmella laitteella tapahtuu laiterikko ja huollon kestoksi on määritelty 100 sekuntia.



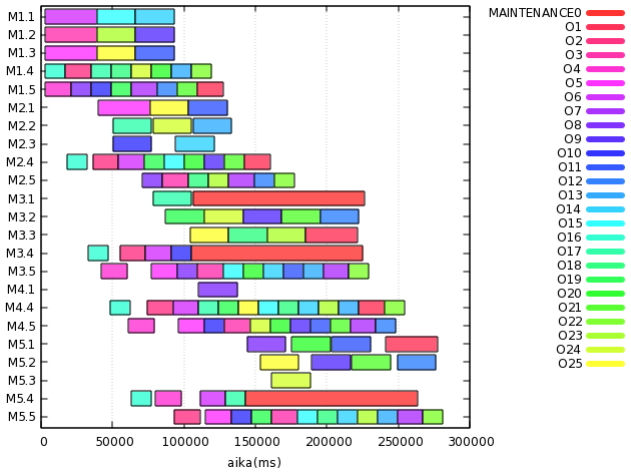

Kuva 17: Esimerkki laiterikosta simulaatiossa.

Suorituskykyvertailu geneettisen algoritmin ja Contract Net -simulaation välillä ajettiin $FH5Q5||C_{max}$ ongelmalla. Laiterikon pituudeksi määritettiin 120 sekuntia ja simulaatio ajettiin laiterikkojen määrillä 5 ja 10. Laiterikot arvottiin tasaisesti kaikkien laitteiden välillä ja jokaiselle rikkoontuvalle laitteelle arvottiin operaation järjestysnumero jolla laite rikkoontuu. Näin ollen jokaisessa simulaatiossa ei välttämättä saavutettu laiterikkojen maksimia, jos tilaukset jakaantuivat epätasaisesti

eri laitteiden välille. Laiterikkojen simuloinnin mittaustulokset näkyvät taulukossa 10, jossa $breaks_{max}$ viittaa laiterikkojen maksimimäärään simulaatiossa. Kuvassa 18 on esimerkki geneettisen algoritmin tuottamasta aikataulusta ongelmalla $FH5Q5|breaks_{max} = 5|C_{max}$ jossa kolme laiterikkoa on realisoitunut.

| Ongelma | $n$ | Tulos | Contract Net ($ms$) | GA ($ms$) | $ContractNet/GA$ |
|---|---|---|---|---|---|
| $FH5Q5\|breaks_{max} = 0\|C_{max}$ | 25 | MEDIAANI | 289005 | 239474 | 1,21 |
| $FH5Q5\|breaks_{max} = 0\|C_{max}$ | 25 | KA | 287961 | 240022 | 1,20 |
| $FH5Q5\|breaks_{max} = 0\|C_{max}$ | 25 | MIN | 275483 | 233464 | 1,18 |
| $FH5Q5\|breaks_{max} = 0\|C_{max}$ | 25 | MAX | 306522 | 250494 | 1,22 |
| $FH5Q5\|breaks_{max} = 5\|C_{max}$ | 25 | MEDIAANI | 301530 | 274033 | 1,10 |
| $FH5Q5\|breaks_{max} = 5\|C_{max}$ | 25 | KA | 309278 | 277274 | 1,12 |
| $FH5Q5\|breaks_{max} = 5\|C_{max}$ | 25 | MIN | 286470 | 246489 | 1,16 |
| $FH5Q5\|breaks_{max} = 5\|C_{max}$ | 25 | MAX | 389631 | 359640 | 1,08 |
| $FH5Q5\|breaks_{max} = 10\|C_{max}$ | 25 | MEDIAANI | 331063 | 316580 | 1,05 |
| $FH5Q5\|breaks_{max} = 10\|C_{max}$ | 25 | KA | 332911 | 328594 | 1,01 |
| $FH5Q5\|breaks_{max} = 10\|C_{max}$ | 25 | MIN | 295530 | 285513 | 1,04 |
| $FH5Q5\|breaks_{max} = 10\|C_{max}$ | 25 | MAX | 411675 | 396676 | 1,04 |

Taulukko 10: Suorituskykyvertailu Hybrid Flow Shop -ongelmalla ja laiterikkojen simuloinnilla




Kuva 18: Esimerkki aikataulusta: geneettinen algoritmi $FH5Q5|breaks_{max} = 5|C_{max}$.

## 5.5 Koetulosten analyysi

### 5.5.1 Triviaali tapaus

Kahden koneen tapauksesta nähdään, että edes yksinkertaisimmassa tapauksessa puhdas markkinamekanismi ei takaa optimaalista suorituskykyä. Syy, miksi joissain tilanteissa kaikki tilaukset menevät ensimmäiselle laiteagentille, johtuu testiasetelmasta ja tilausten käsittelyjärjestyksestä. Testiasetelmassa ensimmäisen agentin suorituskyky on kaksinkertainen verrattuna toiseen agenttiin ja tilausten koot ovat kertoimeltaan 1, 2 ja 4. Jos pienin kertoimen 1 tilaus otetaan suoritukseen ensimmäiselle agentille, on jäljellä olevien tilausten valmistumisaika toisella agentilla (4 ja 8 yksikköä) aina vähintään yhtä suuri kuin ensimmäisellä (3 ja 7 yksikköä) mikäli tilaukset huutokaupataan käänteisessä suuruusjärjestyksessä.

## 5.6 Identtiset tilaukset

Reiät identtisten tilausten tapauksessa johtuvat myös markkinamekanismista. Koska useampi tilausagentti voi saada laiteagentilta saman tarjouksen ja hyväksyä saamansa tarjouksen eriaikaisesti, voi useampi tilaus joutua saman laiteagentin työjonoon kuin olisi optimaalista.

### 5.6.1 Suorituskykyvertailu Hybrid Flow Shop -ongelmalla

Taulukossa 9 olevien tulosten perusteella *geneettinen algoritmi* tuottaa käytännössä aina parempia tuloksia kuin *Contract Net -simulaatio*. Vain yksinkertaisimmassa tapauksessa molempien algoritmien suorituskyky on samankaltainen ja molemmat algoritmit löytävät pääasiassa optimaalisen suunnitelman. Huomioitavaa on, että Contract Net -simulaation pahin tapaus on kuitenkin geneettistä algoritmia huonompi.

Verrattaessa esimerkkejä aikatauluista (15 ja 16) Contract Net -simulaation ongelmana vaikuttaisi olevan tilausten kasaantuminen liiaksi nopeammille laitteille simulaation viimeisimmissä vaiheissa. Kuvassa 15 olevassa aikataulussa ei vaiheista 4 ja 5 käytetä kuin kahta laitetta per vaihe. Geneettinen algoritmi hyödyntää myös hitaampia laitteita näissä vaiheissa, kuten näkyy kuvasta 16.

Lisäksi mittaustulokset sisältävät simulaation asetusajat sekä optimoinnin vaatiman ajan, joten suorituskyvyn perusteella geneettinen algoritmi on aina suorituskyvyltään yhtä hyvä tai parempi kuin Contract Net. Contract Net -algoritmin eduksi jää luotettavuus, koska yhden tilausagentin kaatuminen ei vaikuta muiden tilausagenttien suoritukseen. Optimointiagentin kaatuminen taas pysäyttäisi tuotannon kokonaan.

### 5.6.2 Laiterikkojen simulointi

Taulukosta 10 havaitaan eri algoritmien suorituskyvyn lähestyvän toisiaan laiterikkojen määrän lisääntyessä: $FH5Q5||C_{max}$ ongelmalla algoritmien suorituskyky on lähes identtinen kun laiterikkoja tapahtuu maksimissaan 10 simulaation aikana.

Molempien algoritmien suoritusaika kasvaa laiterikkojen määrän lisääntyessä, koska laiterikot syövät konekapasiteettia. *Geneettinen algoritmi* kuitenkin hidastuu enemmän, mikä johtuu siitä, että geneettinen algoritmi muuttuu käytännössä *Contract Net-algoritmia* vastaavaksi, jos suorittavaa laitetta ei ole saatavilla. Tällöin toteutettu ratkaisu allokoi tilauksen laitteelle jolla ennustettu valmistumisaika on pienin, käytännössä siis Contract Net -algoritmin lopputulos ilman neuvottelua.

# 6 Johtopäätökset

Työn tavoitteena oli tutkia agenttipohjaisia menetelmiä MES-näkökulmasta. Varsinaisia ulkopuolisia vaatimuksia työlle ei ollut, koska työ on tehty oman kiinnostuksen pohjalta eikä esimerkiksi yritykselle. Tavoitteet tulivat siinä mielessä täytetyiksi, että työn lopputuloksena saatiin aikaiseksi toimiva agenttipohjainen tuotantosimulaatio ja kokemuksia agenttipohjaisen järjestelmän toteutukseen liittyvistä haasteista.

Aihealue, varsinkin tuotannon hienokuormitus, on yksi tutkituimmista aihealueista ja materiaalin suhteen työssä oli runsaudenpula. Valtavasta määrästä aiheeseen liittyvää materiaalia oli haastavaa löytää työn kannalta oleellisimmat lähteet. Alan perustiedot ovat myöskin pääosin oppikirjamateriaalia, joten aika suuri osa lähteistä oli oppikirjoja. Taustatieto- ja hienokuormituskappaleissa on siis ennemmin yleisluontoinen kuvaus aihealueesta kuin kaiken kattava. Mikäli työssä olisi ollut asiakaslähtöinen ongelma, olisi myöskin lähdemateriaalia ollut helpompi painottaa juuri tiettyyn ongelmaan.

Tärkein saavutus opinnäytetyössä oli JADE-alustalla toteutetun moniagenttijärjestelmän toteutus. Toteutuksessa käytettiin agenttialustan standardeja ominaisuuksia sekä tutustuttiin FIPA-standardeihin, joita JADE-alusta tukee. Varsinainen malli markkinapohjaisen ratkaisun toteutukseen on hyvin samankaltainen eri lähteissä[28, p. 705][31, s. 408], joten ongelman mallintaminen agentteina oli ennemmin standardiratkaisun hyväksikäyttöä kuin täysin uuden ratkaisun keksimistä ongelmaan.

Varsinaiset haasteet toteutuksessa eivät tulleetkaan itse ratkaisun mallintamisesta, vaan implementoinnissa JADE-alustalla. Yksi esimerkki näistä toteutuksessa havaituista haasteista oli Contract Net -protokollan ajautuminen lukkiumaan, kun tarjouspyyntöjen käsittelijöitä oli vähemmän kuin itse tarjouspyyntöjä. Tässä tapauksessa lukkiuma aiheutui ristikkäisestä odotuksesta, vaikka jokainen järjestelmän komponentti oli erillinen ja kommunikoi toistensa kanssa asynkronisten viestinlähetyksen avulla. Käytännössä vaikutus ohjelmistokehitykseen on näiltä osin samankaltainen rinnakkaisohjelmoinnin kanssa, eli vaikka ratkaisu on käsitteellisellä tasolla helppo ymmärtää, niin toteutuksen yksityiskohdat on vaikeampi saada virheettömiksi kuin esim. yksisäikeisessä/synkronisessa/ei hajautetussa ratkaisussa. Huomioitavaa myös on, että esimerkiksi $FH10Q10||C_{max}$ $n = 100$ -ongelman tapauksessa syntyvässä järjestelmässä on yli 200 agenttia ja säiettä, jolloin järjestelmän debuggaus on erittäin hankalaa ongelmatilanteissa.

Jos mietitään, miten agenttipohjaista ratkaisua voisi laajentaa muihin MES-toiminnallisuuksiin kuin hienokuormitukseen, niin tiedonkeruu olisi suhteellisen helppo lisätä agentteihin. Toteutetussa ratkaisussa agentit lähettävät tilatietoja simulaatioagentille, mutta aivan vastaavasti ne voisivat lähettää viestejä simulaatioagentin sijaan vaikka pilvipalveluun. Tällöin oltaisiin lähellä arkkitehtuuria, joka esitellään artikkelissa "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination"[36].

Toinen laajennuksen näkökulma olisi laajentaa liitettävyyttä muihin tuotannon tietojärjestelmiin ISA-95 standardin mukaisen hierarkian perusteella. Taustatiedot kappaleessa esiteltyjä integraatiotapoja käytettäessä tarvittaisiin agenttipohjaiseen ratkaisuun liitynnät tason 4 järjestelmiin sekä tasojen 1 tai 2 ohjausjärjestelmiin.

Esimerkkinä tarvittaisiin agentti vastaanottamaan tilauksia ERP-järjestelmästä ja luomaan näiden tilausten pohjalta uusia tilausagentteja järjestelmään sekä raportoimaan lopputulokset takaisin. ERP-järjestelmän osalta kyseeseen voisi tulla ratkaisu joka parsisi ERP-järjestelmästä tulevan B2MML-muotoisen sanoman ja rakentaisi oliot agenttijärjestelmään sanoman pohjalta. Vastaavasti alemman tason järjestelmiin kytkeytyminen voitaisiin rakentaa sisään laiteagentteihin, jolloin ne tuotannon simulaation sijaan ohjaisivat alemman tason järjestelmiä käyttäen kommunikointiin esimerkiksi OPC-standardia.

Lisäksi hyviä jatkotutkimuksen aiheita olisi erilaisten hienokuormitusskenaarioiden tarkempi testaus ja analysointi. Nyt käytetty simulaatiomalli ei mm. ota huomioon erilaisia asetusaikoja, välivarastoja, suunniteltuja huoltoja, rajoituksia reitityksessä jne. reaalimaailmassa tuotantoon vaikuttavia tekijöitä. Työssä testattiin kaksi erilaista hienokuormitusratkaisua, hajautettu ja keskitetty optimointi - näiden lisäksi jatkotutkimuksen kohteena voisi olla hajautetut optimointimallit kuten ACO.

Kokeellisessa tutkimuksessa saatiin todennettua kirjallisuudesta löytyneet varaukset agenttipohjaisen hienokuormituksen suorituskyvystä [31, s. 414]. Markkinamekanismilla toimiva agenttipohjainen ratkaisu ei löydä optimaalista ratkaisua aina edes mahdollisimman yksinkertaisessa kahden koneen ongelmassa. Koska viestien saapuminen ja viestien käsittelyjärjestys riippuu eri agenttien suoritusajoista viestien käsittelyssä, on koko järjestelmän suorituskyvyssä välttämättä satunnaisuutta, mikä sitten voi näkyä tarpeettoman huonon ratkaisun valinnassa.

Johdannossa esitettiin seuraavat kysymykset:

1. Mitä etuja agenttipohjaisesta toteutuksesta on?
2. Miten MES järjestelmien ominaisuuksia voitaisiin toteuttaa agenttipohjaisesti?
3. Minkälainen voisi olla tulevaisuuden MES-järjestelmän arkkitehtuuri?

Työn pohjalta voidaan vastata agenttipohjaisen toteutuksen eduiksi uudelleen käytettävyys, koska agentin peruslogiikka on uudelleenkäytettävissä sekä simulaatioettä tuotantoympäristössä. Esimerkiksi rajapintoja lisäämällä voitaisiin opinnäytetyössä toteutettuja agentteja laajentaa toimimaan fyysisten laitteiden ja tietojärjestelmien kanssa. Lisäksi agenttipohjainen järjestelmä havaittiin varsin mukautuvaksi erikoistilanteisiin, vaikka optimaalista suorituskykyä ei aina saavutettu.

Opinnäytetyössä toteutettiin agenttipohjaisesti MES-järjestelmän hienokuormitustoimintoa vastaava osuus. Kuten aiemmissa kappaleissa pohdittiin, olisi toiminnallisuuden laajentaminen muihin alueisiin, kuten tiedonkeräykseen suhteellisen helppoa. MES-järjestelmän osista hankalimmin agenttipohjaisesti toteutettavia lienee raportointiratkaisut, joissa on etua tiedon keräämisestä ja keskittämisessä yhteen paikkaan.

Työn pohjalta kokonaisen MES-järjestelmän arkkitehtuurista voidaan esittää vain arvailuja. Ratkaisu voi olla joko yhdistetty agenttipohjainen ja pilvialustaa hyväksi käyttävä toteutus[36] tai toinen vaihtoehto on, että MES-toiminnallisuutta aletaan ajaa alemman tason järjestelmissä kuten PLC-logiikoissa[5]. Työn puitteissa ei ollut mahdollista vertailla erilaisia agenttialustoja tai erilaisia toteutustapoja eikä

esim. JADE-järjestelmän soveltuvuutta tuotantokäyttöön pysty työn pohjalta arvioimaan. JADE-järjestelmän soveltuvuuden arviointi vaatisi pidempiä yhtäjaksoisia suoritusajoja sekä kuormitustestausta.

Lopputuloksena työssä saatiin toteutetuksi ja tutkituksi agenttipohjaisia menetelmiä MES-näkökulmasta ja saatiin ideoita jatkotutkimusta varten.

# Viitteet

[1] *ANSI/ISA–88.01–1995 Batch Control Part 1: Models and Terminology*. International Society of Automation (ISA), lokakuu 1995.

[2] *ANSI/ISA-95.00.01-2010 (IEC 62264-1 Mod) Enterprise-Control System Integration - Part 1: Models and Terminology*. International Society of Automation (ISA), toukokuu 2010.

[3] *ANSI/ISA-95.00.02-2010 (IEC 62264-2 Mod) Enterprise-Control System Integration - Part 2: Object Model Attributes*. International Society of Automation (ISA), toukokuu 2010.

[4] *ANSI/ISA-95.00.03-2013 (IEC 62264-3 Modified) Enterprise-Control System Integration - Part 3: Activity Models of Manufacturing Operations Management*. International Society of Automation (ISA), heinäkuu 2013.

[5] N. Antzoulatos et al. "Interfacing Agents with an Industrial Assembly System for "Plug and Produce": (Demonstration)". Teoksessa: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '15. Istanbul, Turkey: International Foundation for Autonomous Agents ja Multiagent Systems, 2015, s. 1957–1958. ISBN: 978-1-4503-3413-6. URL: http://dl.acm.org/citation.cfm?id=2772879.2773523.

[6] M. E. Aydin ja T. C. Fogarty. "A simulated annealing algorithm for multi-agent systems: a job-shop scheduling application". English. *Journal of Intelligent Manufacturing* 15.6 (joulukuu 2004). Copyright - Copyright (c) 2004 Kluwer Academic Publishers; Document feature - references; tables; graphs; equations; diagrams; Last updated - 2012-07-24, s. 805–814. URL: https://search.proquest.com/docview/200508340?accountid=27468.

[7] F. L. Bellifemine, G. Caire ja D. Greenwood. *Developing Multi-Agent Systems with JADE*. Chichester, UK: Wiley, 2007. ISBN: 978-0-470-05747-6.

[8] P. Brucker, B. Jurisch ja B. Sievers. "A branch and bound algorithm for the job-shop scheduling problem". *Discrete Applied Mathematics* 49.1 (1994). Special Volume Viewpoints on Optimization, s. 107–127. ISSN: 0166-218X. DOI: https://doi.org/10.1016/0166-218X(94)90204-6. URL: http://www.sciencedirect.com/science/article/pii/0166218X94902046.

[9] J. Cadavid et al. "Conceiving the Model-driven Smart Factory". Teoksessa: *Proceedings of the 2015 International Conference on Software and System Process*. ICSSP 2015. Tallinn, Estonia: ACM, 2015, s. 72–76. ISBN: 978-1-4503-3346-7. DOI: https://doi.org/10.1145/2785592.2785602.

[10] M. Cossentino et al. "Metrics for Evaluating Modularity and Extensibility in HMAS Systems". Teoksessa: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '15. Istanbul, Turkey: International Foundation for Autonomous Agents ja Multiagent Systems, 2015, s. 1061–1069. ISBN: 978-1-4503-3413-6. URL: http://dl.acm.org/citation.cfm?id=2772879.2773286.

[11] *FIPA Abstract Architecture Specification*. Component. Foundation for Intelligent Physical Agents, joulukuu 2002. URL: http://www.fipa.org/specs/fipa00001/SC00001L.pdf (viitattu 14. 11. 2017).

[12] *FIPA Communicative Act Library Specification*. Component. Foundation for Intelligent Physical Agents, joulukuu 2002. URL: http://www.fipa.org/specs/fipa00037/SC00037J.pdf (viitattu 14. 11. 2017).

[13] *FIPA Contract Net Interaction Protocol Specification*. Component. Joulukuu 2002. URL: http://www.fipa.org/specs/fipa00029/SC00029H.pdf (viitattu 14. 11. 2017).

[14] M. R. Garey ja D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN: 0716710455.

[15] *Gazebo (LA-CC-13-049) - util / Gantt*. Los Alamos National Lab. URL: https://github.com/lanl/Gazebo/tree/master/bin/util/Gantt (viitattu 14. 11. 2017).

[16] F. Geyik ja H. C. Ismail. "The strategies and parameters of tabu search for job-shop scheduling". English. *Journal of Intelligent Manufacturing* 15.4 (elokuu 2004). Copyright - Copyright (c) 2004 Kluwer Academic Publishers; Document feature - references; tables; equations; graphs; Last updated - 2012-07-24, s. 439–448. URL: https://search.proquest.com/docview/200522988?accountid=27468.

[17] *ISA100 Wireless Technical Overview Brochure*. ISA100 Wireless Compliance Institute. URL: https://isa100wci.org/en-US/Documents/PDF/3405-ISA100-WirelessSystems-Future-broch-WEB-ETSI.aspx (viitattu 14. 11. 2017).

[18] *JAVA Agent DEvelopment Framework*. Telecom Italia S.p.A. URL: http://jade.tilab.com/ (viitattu 14. 11. 2017).

[19] K. Kravari ja N. Bassiliades. "A Survey of Agent Platforms". *Journal of Artificial Societies and Social Simulation* 18.1 (2015), s. 11. ISSN: 1460-7425. DOI: https://doi.org/10.18564/jasss.2661. URL: http://jasss.soc.surrey.ac.uk/18/1/11.html.

[20] J. B. H. Kwa. "Tolerant Planning and Negotiation in Generating Coordinated Movement Plans in an Automated Factory". Teoksessa: *Proceedings of the 1st International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems - Volume 1*. IEA/AIE '88. Tullahoma, Tennessee, USA: ACM, 1988, s. 522–529. ISBN: 0-89791-271-3. DOI: https://doi.org/10.1145/51909.51969.

[21] H. Lasi et al. "Industry 4.0". *Business & Information Systems Engineering* 6.4 (2014), s. 239–242. ISSN: 1867-0202. DOI: https://doi.org/10.1007/s12599-014-0334-4.

[22] P. Leitão. "Agent-based distributed manufacturing control: A state-of-the-art survey". *Engineering Applications of Artificial Intelligence* 22.7 (2009). Distributed Control of Production Systems, s. 979–991. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2008.09.005. URL: http://www.sciencedirect.com/science/article/pii/S0952197608001437.

[23] G. Y. Lin ja J. J. Solberg. "An Agent-based Flexible Routing Manufacturing Control Simulation System". Teoksessa: *Proceedings of the 26th Conference on Winter Simulation*. WSC '94. Orlando, Florida, USA: Society for Computer Simulation International, 1994, s. 970–977. ISBN: 0-7803-2109-X. URL: http://dl.acm.org/citation.cfm?id=193201.194638.

[24] A. Madureira, F. Santos ja I. Pereira. "Self-managing Agents for Dynamic Scheduling in Manufacturing". Teoksessa: *Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation*. GECCO '08. Atlanta, GA, USA: ACM, 2008, s. 2187–2192. ISBN: 978-1-60558-131-6. DOI: https://doi.org/10.1145/1388969.1389045.

[25] V. Mařík, P. Vrba ja M. Fletcher. "Agent-Based Simulation: Mast Case Study". Teoksessa: *Emerging Solutions for Future Manufacturing Systems: IFP TC 5 / WG 5.5 Sixth IFIP International Conference on Information Technology for Balanced Automation Systems in Manufacturing and Services 27–29 September 2004, Vienna, Austria*. Toim. L. M. Camarinha-Matos. Boston, MA: Springer US, 2005, s. 61–72. ISBN: 978-0-387-22829-7. DOI: https://doi.org/10.1007/0-387-22829-2_7.

[26] A. Márkus, T. K. Váncza ja L. Monostori. "A Market Approach to Holonic Manufacturing". *CIRP Annals* 45.1 (1996), s. 433–436. ISSN: 0007-8506. DOI: https://doi.org/10.1016/S0007-8506(07)63096-0. URL: http://www.sciencedirect.com/science/article/pii/S0007850607630960.

[27] H. Meyer, F. Fuchs ja K. Thiel. *Manufacturing Execution Systems: Optimal Design, Planning, and Deployment*. Toim. H. Meyer. Mc Graw Hill Education, 2009.

[28] L. Monostori, J. Váncza ja S. Kumara. "Agent-Based Systems for Manufacturing". *{CIRP} Annals - Manufacturing Technology* 55.2 (2006), s. 697–720. ISSN: 0007-8506. DOI: https://doi.org/10.1016/j.cirp.2006.10.004. URL: http://www.sciencedirect.com/science/article/pii/S1660277306000053.

[29] S. S. Panwalkar ja W. Iskander. "A Survey of Scheduling Rules". *Oper. Res.* 25.1 (helmikuu 1977), s. 45–61. ISSN: 0030-364X. DOI: https://doi.org/10.1287/opre.25.1.45.

[30] M. Pěchouček, M. Rehák ja V. Marik. "Expectations and Deployment of Agent Technology in Manufacturing and Defence: Case Studies". Teoksessa: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS '05. The Netherlands: ACM, 2005, s. 100–106. ISBN: 1-59593-093-0. DOI: https://doi.org/10.1145/1082473.1082811.

[31] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. 3rd. Springer Publishing Company, Incorporated, 2008. ISBN: 978-0-387-78935-4.

[32] S. Poslad. "Specifying Protocols for Multi-agent Systems Interaction". *ACM Trans. Auton. Adapt. Syst.* 2.4 (marraskuu 2007). ISSN: 1556-4665. DOI: https://doi.org/10.1145/1293731.1293735.

[33] R. Ruiz ja J. A. Vázquez-Rodríguez. "The hybrid flow shop scheduling problem". *European Journal of Operational Research* 205.1 (2010), s. 1–18. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2009.09.024.

[34] B. Scholten. *MES Guide for Executives: Why and how to Select, Implement, and Maintain a Manufacturing Execution System*. International Society of Automation, 2009. ISBN: 9781936007035.

[35] W. Shen et al. "Applications of agent-based systems in intelligent manufacturing: An updated review". *Advanced Engineering Informatics* 20.4 (2006), s. 415–431. ISSN: 1474-0346. DOI: https://doi.org/10.1016/j.aei.2006.05.004. URL: http://www.sciencedirect.com/science/article/pii/S1474034606000292.

[36] S. Wang et al. "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination". *Computer Networks* 101.Supplement C (2016). Industrial Technologies and Applications for the Internet of Things, s. 158–168. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2015.12.017. URL: http://www.sciencedirect.com/science/article/pii/S1389128615005046.

[37] G. Weiss, toim. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. ISBN: 0-262-23203-0.

[38] M. Wooldridge ja N. R. Jennings. "Intelligent Agents: Theory and Practice". *Knowledge Engineering Review* 10 (1995), s. 115–152.

[39] W. Xiao et al. "Hybrid flow shop scheduling using genetic algorithms". Teoksessa: *Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No.00EX393)*. Vol. 1. 2000, 537–541 vol.1. DOI: https://doi.org/10.1109/WCICA.2000.860026.

# A B2MML esimerkki

Tässä liitteessä on esimerkki B2MML-muotoisesta tilauksesta ja tilaukseen lähetetystä vastineesta. Huomioitavaa rakenteessa on tilauksen (1. esimerkki) sisältämät vaatimukset laitteille sekä tuotettavalle materiaalille. Tilauksen valmistumisesta kertovassa vastineessa vastaavasti kerrotaan käytetty laite ja tuotettu materiaali määrineen.

B2MML-muotoinen tilaus (Production Request):

```
<?xml version="1.0"?>
<ProductionSchedule xmlns="http://www.wbf.org/xml/b2mml-v02">
  <ID>Default</ID>
  <ProductionRequest>
    <ID>20171116001</ID>
    <Description>HEL1</Description>
    <ProductProductionRuleID>100000_EX1</ProductProductionRuleID>
    <StartTime>2017-11-16T00:00:00</StartTime>
    <EndTime>2017-11-16T07:00:00</EndTime>
    <SegmentRequirement>
      <ID>0001</ID>
      <ProcessSegmentID>0001</ProcessSegmentID>
      <Description>Packing</Description>
      <Duration>2H</Duration>
      <EquipmentRequirement>
        <EquipmentID>WC1</EquipmentID>
        <Quantity>
          <QuantityString>1</QuantityString>
          <DataType>float</DataType>
          <UnitOfMeasure>n/a</UnitOfMeasure>
          <Any/>
        </Quantity>
        <Any/>
      </EquipmentRequirement>
      <MaterialProducedRequirement>
        <MaterialClassID/>
        <MaterialDefinitionID>100000</MaterialDefinitionID>
        <Location>
          <EquipmentID>0000</EquipmentID>
          <EquipmentElementLevel>Site</EquipmentElementLevel>
          <Location>
            <EquipmentID>0001</EquipmentID>
            <EquipmentElementLevel>Area</EquipmentElementLevel>
          </Location>
        </Location>
        <Quantity>
          <QuantityString>500</QuantityString>
          <DataType>float</DataType>
          <UnitOfMeasure>KG</UnitOfMeasure>
          <Any/>
        </Quantity>
        <Any/>
      </MaterialProducedRequirement>
      <Any/>
    </SegmentRequirement>
    <Any/>
  </ProductionRequest>
  <Any/>
</ProductionSchedule>
```

B2MML-muotoinen tilausvahvistus (Production Response):

```
<?xml version="1.0"?>
<ProductionPerformance xmlns="http://www.wbf.org/xml/b2mml-v02">
  <ID>1</ID>
  <Description />
  <PublishedDate>2017-11-16T02:02:02</PublishedDate>
  <ProductionResponse>
    <ID>20171116001</ID>
    <ProductProductionRuleID />
    <SegmentResponse>
      <ID>0001</ID>
      <Description>1</Description>
      <ActualStartTime>2017-11-16T00:01:00</ActualStartTime>
      <ProductionData>
        <ID>DURATION</ID>
        <Value>
          <ValueString>2.2</ValueString>
          <DataType>float</DataType>
          <UnitOfMeasure>H</UnitOfMeasure>
          <Any />
        </Value>
      </ProductionData>
      <EquipmentActual>
        <EquipmentID>WC1</EquipmentID>
        <Description />
        <Location>
          <EquipmentID />
          <EquipmentElementLevel>Enterprise</EquipmentElementLevel>
          <Location>
            <EquipmentID />
            <EquipmentElementLevel>Enterprise</EquipmentElementLevel>
            <Any />
          </Location>
          <Any />
        </Location>
        <Any />
      </EquipmentActual>
      <MaterialProducedActual>
        <MaterialDefinitionID>100000</MaterialDefinitionID>
        <MaterialLotID>1611017</MaterialLotID>
        <MaterialSubLotID>201716110001</MaterialSubLotID>
        <Description>1</Description>
        <Location>
          <EquipmentID>0000</EquipmentID>
          <EquipmentElementLevel>Site</EquipmentElementLevel>
          <Location>
            <EquipmentID>0001</EquipmentID>
            <EquipmentElementLevel>Area</EquipmentElementLevel>
          </Location>
        </Location>
        <Quantity>
          <QuantityString>500.0</QuantityString>
          <DataType>float</DataType>
          <UnitOfMeasure>KG</UnitOfMeasure>
        </Quantity>
        <Any>
          <Scrap>
            <QuantityString>0.0</QuantityString>
            <DataType>float</DataType>
            <UnitOfMeasure>KG</UnitOfMeasure>
          </Scrap>
        </Any>
      </MaterialProducedActual>
      <Any>
        <ERP_WORKCENTER>WC1</ERP_WORKCENTER>
        <PlantID>0000</PlantID>
      </Any>
    </SegmentResponse>
    <Any>
      <PlantID>0000</PlantID>
      <Type>Production Order</Type>
    </Any>
  </ProductionResponse>
</ProductionPerformance>
```

# B Mittausten suoritus

Tässä liitteessä kuvataan kuinka mittausajot suoritettiin algoritmien vertailua varten.

*Simulaatioagentti* ottaa parametreikseen seuraavat tiedot numeroa vastaavassa järjestyksessä:

1. tiedoston nimi
2. vaiheiden määrä
3. laitteiden määrä per vaihe
4. tilausten määrä
5. laiteluokkien määrä
6. laitteiden peruskilpinopeus
7. laitteita 1. laiteluokassa
8. nopeuskerroin luokkien välillä
9. tilausluokkien määrä
10. tilauksien peruskoko
11. tilausten 1. tilausluokassa
12. tilausten kokokerroin luokkien välillä
13. optimointi (true = GA / false = Contract Net)
14. laiterikkojen määrä
15. laiterikon pituus sekunteina

Simulaatioagentin ja JADE:n käynnistys parametreineen (*start.sh*):

```
#!/bin/sh
/usr/lib/jvm/java-1.8.0-openjdk/jre/bin/java -classpath "/home/mrahikai/eclipse-workspace/agent-scheduling-sim/build/libs/*" \
jade.Boot -gui "simulation_controller:net.mikkorahikainen.agentschedulingsim.agents.SimulationAgent(\$1, \$2, \$3, \$4, \$5, \$6, \$7, \$8, \$9, \${10}, \${11}, \${12}, \${13}, \${14}, \${15})"
```

Jokaista skenaariota ajettiin bash-skriptin avulla 20 kertaa. Bash-skripti suoritti vuorotellen Contract Net -simulaatiota sekä geneettistä algoritmia, jolloin vaikutukset koneen muusta kuormituksesta jakaantuivat myös ajallisesti tasaisesti algoritmien välillä.

Käytetty bash-skripti:

```
#!/bin/bash
for i in {1..20}
do
./start.sh "sim_5_5/test_\$i.txt" 5 5 25 2 10000 2 2 2 100 6 0.75 false 0 0
./start.sh "sim_5_5/testopt_\$i.txt" 5 5 25 2 10000 2 2 2 100 6 0.75 true 0 0
done
```

Lopputuloksena syntyy joukko tiedostoja, joiden formaattina on tekstimuotoinen esitys Gantt-kaaviosta:

```
M1.2    1045     37103     O2
M1.1    2036     38094     O1
M1.2    37107    64148     O3
M1.1    38096    65138     O4
M2.2    38079    74133     O2
M2.1    39078    75137     O1
M2.1    75139    102185    O3
M2.2    76139    103185    O4
```

Gantt-kaaviot suoritustuloksista muodostettiin python-skriptillä *gantt.py*[15], joka tuottaa lopputuloksenaan komentosarjat *gnuplot*-ohjelmalle.

```
python gantt.py -o sim55_test_2.gpl sim_5_5_5b/test_2.txt
```